Modes of Operations for Encryption and Authentication Using Stream Ciphers Supporting an Initialisation Vector

Palash Sarkar

Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road, Kolkata
India 700108.
email: palash@isical.ac.in

Abstract. We describe a systematic framework for using a stream cipher supporting an initialisation vector (IV) to perform various tasks of authentication and authenticated encryption. These include message authentication code (MAC), authenticated encryption (AE), authenticated encryption with associated data (AEAD) and deterministic authenticated encryption (DAE) with associated data. Several schemes are presented and rigourously analysed. A major component of the constructions is a keyed hash function having provably low collision and differential probabilities. Methods are described to efficiently extend such hash functions to take multiple inputs. In particular, double-input hash functions are required for the construction of AEAD schemes. An important practical aspect of our work is that a designer can combine off-the-shelf stream ciphers with off-the-shelf hash functions to obtain secure primitives for MAC, AE, AEAD and DAE(AD).
Keywords: stream cipher with IV, universal hash function, authentication, authenticated encryption with associated data, deterministic authenticated encryption, modes of operations.

1 Introduction

Stream ciphers are one of the basic primitives for performing cryptographic operations. In the conventional view, a stream cipher is considered to be a pseudo-random generator. It takes as input a short secret random seed (called the secret key) and produces as output a long random looking keystream. Encryption is performed by bitwise XORing the keystream to the message. Decryption is performed by regenerating the keystream (from the shared secret seed) and XORing it to the ciphertext.

Stream ciphers defined in the above manner are not very convenient for applications. A single secret key cannot be securely used to encrypt two different messages. So, for each message, it is required to have a separate secret key. As a result, the key management issue becomes complicated.

In the modern view, a stream cipher SC takes as input a secret key K and an initialisation vector (IV). One of the early works which advocated this approach was [32] and all proposals in the estream [2] portfolio support an IV. The IV is not required to be secret. A message M is encrypted as $M \oplus \mathsf{SC}_K(\mathsf{IV})$. The IV is communicated to the receiver publicly, or could be generated at both ends as a counter value. Flexibility in usage arises from the fact that the same key can now be used with different messages; the IV only needs to be changed. Since there is no secrecy requirement on the IV, this is a easier task to manage.

This view of stream ciphers has, however, changed the theoretical model of a stream cipher as a pseudo-random generator. The keystream that is used to encrypt a particular message, is still required to appear random to a computationally bounded adversary. So, the requirement is that for distinct values of IV, the outputs $\mathsf{SC}_K(\mathsf{IV})$ should appear independent and uniform random to

a computationally bounded adversary. This is theoretically captured by the notion of a pseudo-random function (PRF), and in the modern view, a stream cipher with IV is modelled as a PRF. See [6] for further justification of this view.

The starting point of our work is the model of a stream cipher with IV as a PRF. This is a PRF which takes as input a short fixed-length string and produces as output a long string. We consider the problem of constructing higher-level primitives using such a PRF. The primitives that we consider are message authentication code (MAC), authenticated encryption (AE), AE with associated data (AEAD) and deterministic authenticated encryption (DAE) with associated data. For each of these primitives, we describe efficient constructions and provide formal security analyses.

All of the primitives that we consider have important practical applications. Efficient methods to construct them are required to actually deploy cryptographic mechanisms. Previous work has mostly considered methods to construct these primitives based on block ciphers. Our work is the first systematic work showing how to obtain these primitives from stream ciphers supporting an IV.

A core requirement in all the constructions is a keyed hash function which is selected from a family $\{\mathsf{Hash}_\tau\}$ by choosing a uniform random τ. For all distinct x and x' and any y, we require the probability that $\mathsf{Hash}_\tau(x) \oplus \mathsf{Hash}_\tau(x')$ equals y to be low. (We are assuming that Hash_τ produces a fixed-length bit string as output). Such hash functions are called almost XOR universal hash functions. Starting from [12, 43, 17], constructions and efficient implementations of such functions have been studied extensively in the literature (see for example [41, 19, 11, 8]).

For the construction of AEAD schemes, we need an extended notion. The above kind of hash functions take a single input. To handle associated data, the hash function needs to take two inputs. A double-input hash function cannot be naively constructed from a single-input hash function by merely concatenating the two inputs. This is because of the fact that even if (X, Y) is not equal to (X', Y'), it may still be the case that $X||Y = X'||Y'$. Applying a single-input hash function to the concatenated value will lead to the same output even though the pairs (X, Y) and (X', Y') are distinct.

Methods are required for extending a single-input hash function to a multi-input hash function. Iwata and Yasuda [20] have provided a method to handle multiple inputs which work only with polynomial based hashing. Our work, on the other hand, describes several generic methods which cover all single-input hash functions.

As mentioned earlier, the constructions that we describe are important from an application point of view. There are several well known stream ciphers and hash functions. SNOW [15] is a well-known example. The eSTREAM project [2] has resulted in stream ciphers geared either towards fast software implementations or for small hardware implementations. Combining a fast software oriented stream cipher with a fast software oriented hash function using one of the methods proposed in this work will result in a secure primitive with a fast software implementation. Similarly, one can combine a hardware oriented stream cipher with a hardware oriented hash function using our methods to get a secure primitive with small hardware footprint. So, in a sense, our constructions are general templates for obtaining stream cipher based primitives for MAC, AE, AEAD and DAE(AD).

Prior and related work. The literature contains several scattered works on using a stream cipher for MAC and AE. For MAC, the basic idea goes back to [43].

The combined primitive of authenticated encryption was formalised in [22, 4]. Separate definitions of privacy and authenticity of an encryption scheme were given in [5]. An explicit case for formalisation of nonce-based symmetric encryption has been done in [30].

Modes of operations of block ciphers to achieve AE have been studied in the literature and several schemes are known [21, 18, 31, 29, 13, 38]. NIST of USA has standardised one such scheme, called GCM [24, 14]. The issue of authenticating an associated data was first formally tackled in [28]. Later works such as [29, 24, 39] incorporated methods to perform AE and at the same time authenticate an associated data. Deterministic authenticated encryption (with associated data) was formalised in [33] to model the so-called key-wrap problem. Existing works on provable security treatment of AE, AEAD and DAE(AD) schemes have almost exclusively focussed on modes of operations of block ciphers to achieve these functionalities.

A method for achieving AE using a stream cipher is given in [7]. This is one of the several constructions that we define. As far as we are aware of, there has been no previous work on constructing AEAD schemes from a stream cipher with IV. Similarly, there has been no previous work on constructing DAE(AD) schemes from a stream cipher with IV. Stream cipher based constructions for AE have been proposed as a combined primitive (see for example HELIX [16] and its later version PHELIX) but have not been very successful. HELIX has been cryptanalysed in [25]. A recent proposal [3], though, provides a combined AE primitive and it remains to be seen how well it withstands future cryptanalysis.

We would like to point out the differences of our AE constructions with the generic constructions for AE in [4]. The schemes in [4] provide generic methods to combine a symmetric encryption scheme (typically a block cipher mode of operation) and a MAC scheme to obtain AE schemes. In contrast, we provide methods for combining a stream cipher with IV with suitable hash functions with provable differential properties to obtain AE (and other) schemes. Note that the use of MAC schemes in [4] is a stronger requirement than the hash functions that we use. Further, the work [4] pre-dates the notions of AEAD and DAE(AD) and these as well as MAC schemes are not considered in [4]. There are other differences in the overall approach of our work and that of [4]. We carefully consider important practical issues such as obtaining the possibly long keys of the hash function and reducing the number of initialisations of the stream cipher. Such issues are absent in [4] which is primarily a theoretical work.

Importance of the present work: We start by saying what is not a contribution of this work. This paper makes no definitional contributions. As mentioned above, the formal notions of the cryptographic functionalities of authentication, AEAD and DAEAD have already been defined in several important works in the literature. Constructions of the respective functionalities have, however, been pre-dominantly from block ciphers.

The current work sets out to achieve a unified and systematic framework for achieving the above mentioned functionalities using a stream cipher with IV. Relevant practical issues such as tackling the size of the hash key, avoiding repeated initialisations of the stream cipher are dealt with carefully. At the same time, the usual provable security guarantees are obtained for the various constructions.

Though stream ciphers are a fundamental primitive in cryptography, by themselves, however, they are hardly useful. To be actually used, a practitioner needs secure and efficient methods to build higher level primitives like MAC, AE(AD) and DAE(AD) schemes from stream ciphers. The literature, on the other hand, hardly covers this important area of research. To fill this gap, this work provides the first systematic treatment of this topic and achieves for stream ciphers what many papers in the literature have together achieved for block ciphers. It will be useful to a practitioner to find a compact description of the constructions of different important cryptographic primitives from stream ciphers. Currently, no work exists in the literature with a similar scope and breadth.

2 Preliminaries

In this section, we introduce a few basic notions. We will use the following notation. Given a binary string S, let $\mathsf{len}(S)$ denote the length of S, i.e., $\mathsf{len}(S)$ is the number of bits in S. Given an integer i with $0 \leq i \leq 2^n - 1$, let $\mathsf{bin}_n(i)$ denote the n-bit binary representation of i.

2.1 Pseudo-Random Function (PRF)

Let $\mathcal{D}$ and $\mathcal{R}$ be finite non-empty sets and f be a random (but, not necessarily, uniform random) function from $\mathcal{D}$ to $\mathcal{R}$. In other words, f is drawn from the set of all functions from $\mathcal{D}$ to $\mathcal{R}$ according to some distribution.

For cryptographic applications, a random function f arises from a function family $\{F_K\}_{K \in \mathcal{K}}$, $F_K : \mathcal{D} \to \mathcal{R}$ where a key K is drawn uniformly at random from the finite set $\mathcal{K}$ and f is set to be equal to F_K. The randomness in f arises from the uniform random choice of K.

A possible distinguisher (also called an adversary) $\mathcal{A}$ is a probabilistic algorithm which is given oracle access to f. It adaptively queries the oracle with inputs $x^{(1)}, \ldots, x^{(q)}$, receiving in return the responses $f(x^{(1)}), \ldots, f(x^{(q)})$. At the end of the interaction, $\mathcal{A}$ outputs a bit. Denote by $\mathcal{A}^f \Rightarrow 1$, the event that this output is 1.

Let f^* be a function chosen uniformly at random from the set of all functions mapping $\mathcal{D}$ to $\mathcal{R}$. Adversary $\mathcal{A}$'s interaction with f^* is defined in a manner similar to the above and denote by $\mathcal{A}^{f^*} \Rightarrow 1$, the event that the output of $\mathcal{A}$ after interaction with f^* is 1. The PRF-advantage of $\mathcal{A}$ in distinguishing f from f^* is defined to be the following.

$$\mathsf{Adv}_f^{\mathrm{prf}}(\mathcal{A}) = \Pr[\mathcal{A}^f \Rightarrow 1] - \Pr[\mathcal{A}^{f^*} \Rightarrow 1]. \tag{1}$$

Informally, the above definition captures the idea that the function f is indistinguishable from a uniform random function f^*. In the context of function families, the above can be viewed in the following manner. A uniform random K is chosen and kept secret from the adversary $\mathcal{A}$. The adversary adaptively queries the function F_K on distinct inputs $x^{(1)}, \ldots, x^{(q)}$ receiving in return the values $F_K(x^{(1)}), \ldots, F_K(x^{(q)})$. These values should "look random" to $\mathcal{A}$.

A measure of the amount of interaction of the adversary with the oracle f is the number of queries q. This measure, however, may not capture the entire picture. The length of the queries could vary. A better measure is the so-called query complexity. The total query complexity σ of $\mathcal{A}$ is defined to be the total number of bits it provides in all the queries. In other words, this is the sum of the lengths of all the individual queries, i.e., $\sigma = \sum_{i=1}^{q} |x^{(i)}|$. The maximum (more precisely, the supremum) of such advantages over all adversaries which run in time t, make q queries and have total query complexity σ is denoted by $\mathsf{Adv}_f^{\mathrm{prf}}(t, q, \sigma)$.

The definition of advantage in (1) does not use the absolute value. Note that for any adversary $\mathcal{A}$ there is another adversary $\mathcal{A}'$ which is the same as $\mathcal{A}$ except that it flips the output of $\mathcal{A}$ to provide its output. So, if the advantage of $\mathcal{A}$ is negative, then the advantage of $\mathcal{A}'$ is positive. Since the resource bounded advantage is defined to be the maximum over all adversaries, this value will always be positive. On the other hand, avoiding absolute values makes it somewhat simpler to work with the inequalities. This approach, in fact, has been used earlier by other authors.

2.2 Hash Functions

Let $\{\mathsf{Hash}_\tau\}$ be a family of keyed hash functions on a common domain and a common range. The key τ is chosen from an appropriate finite set. In this work, we will be taking the range of a hash

function to be the set of all strings of some fixed length. This set forms an additive group under the operation $\oplus$. Two kinds of probabilities [9] are defined for the hash function family.

Collision probability. For all distinct x and x', the collision probability of Hash_τ corresponding to the pair (x, x') is $\Pr[\mathsf{Hash}_\tau(x) = \mathsf{Hash}_\tau(x')]$, where the probability is taken over the uniform random choice of τ.

Differential probability. For all distinct x and x' and any y, the differential probability of Hash_τ corresponding to the triplet (x, x', y) is $\Pr[\mathsf{Hash}_\tau(x) \oplus \mathsf{Hash}_\tau(x') = y]$, where the probability is taken over the uniform random choice of τ.

If all the collision probabilities are bounded above by ε, then Hash_τ is said to be almost universal (ε-AU); if all the differential probabilities are bounded above by ε, then Hash_τ is said to be almost XOR universal (ε-AXU).

We will assume that the input to a hash function is a binary string. Usually hash functions are defined using some algebraic structure. An input binary string is formatted into a sequence of elements belonging to the algebraic structure and the digest is computed. This digest is represented as a fixed length binary string. In its basic or 'raw' form, there are two issues regarding hash functions that we would like to highlight.

1. A hash function need not be defined for all possible input lengths. For example, a hash function may be defined only for binary strings whose lengths are multiples of 128.
2. Collision and differential probabilities of a hash function can be guaranteed to be low only for equal length inputs.

Both the above issues are usually tackled by employing suitable padding techniques. The result is a single-input hash function without any restriction on the input length and which has provably low collision and differential probabilities for any two inputs (without the restriction of their lengths being equal). In this work, we will consider hash function which take a vector of strings as input. Such multi-input hash functions will be constructed from single-input hash functions which have low collision and differential probabilities only for equal length strings.

There are well known examples of both AU and AXU hash families. Univariate polynomials over finite fields give rise to such families. For such hash function families, a key τ usually consists of a fixed number of elements of the underlying finite field and can be represented by a fixed-length bit-string. The basic idea of polynomial-based hashing yields function families which are AU/AXU only for fixed-length inputs. These can be modified to handle variable-length inputs. (See for example [8, 40] for methods which work for polynomial hash functions and [11] for a generic method which works for any hash function.) The output (called a digest) is a short fixed-length string.

Another class of hash families is known. These can also handle variable length inputs and produce short fixed-length digests. But, the key τ for such hash functions can be quite long. A prefix of the key having length equal to the length of the message is actually used. In certain cases, the length of the key-prefix is actually slightly greater than the length of the message to be hashed. Examples of such hash families are known [17, 11, 35]. In particular, UMAC [11] is a well-known example which can be used to perform very fast hashing in software.

One issue about using such hash functions is the key. Since the key can be quite long, it is not practical to store the key. Instead, a stream cipher is used to generate the key. In this work, we will consider in details how such hash functions can be securely and efficiently combined with a stream cipher supporting an IV.

For the sake of convenience, we will distinguish two types of hash functions.

Type-I. For such hash functions, the key length is independent of the message length. Typically, a key will be a short fixed-length bit string.

Type-II. For such hash functions, the number of bits of the hash key required to produce the digest is as long as the message length. In some cases, it is actually slightly longer than the message length. Suppose τ is the hash key required to hash a message M and τ' is any string such that τ is a prefix of τ'. Then, $\mathsf{Hash}_\tau(M) = \mathsf{Hash}_{\tau'}(M)$, i.e., using a key longer than that necessary to hash a message M does not change the digest.

Distribution of $\mathsf{Hash}_\tau(x)$. Given any fixed x in the domain and any fixed y in the range, the probability $\Pr[\mathsf{Hash}_\tau(x) = y]$ is over the uniform random choice of τ. There is no randomness in the choices of x and y. In fact, $\mathsf{Hash}_\tau(x)$ is a random variable which is distributed over the range of the hash function.

As an example, consider $\mathsf{Hash}_\tau(x)$ to be the usual polynomial based hash function over the field $GF(2^n)$. In this case, x is a tuple $(x_0, \ldots, x_t)$, where x_i's are n-bit strings which encode elements of $GF(2^n)$. The definition of polynomial-based Hash_τ is as follows.

$$\mathsf{Hash}_\tau(x_0, \ldots, x_t) = x_0 \oplus \tau x_1 \oplus \cdots \oplus x_t \tau^t.$$

For any fixed y in $GF(2^n)$ and *non-zero* x, the probability that $\mathsf{Hash}_\tau(x) = y$ is at most $t/2^n$ (which follows from the fact that a univariate polynomial of degree t over a field has at most t roots).

For other examples of Hash_τ (such as the multilinear map from [17]), the random variable $\mathsf{Hash}_\tau(x)$ is uniformly distributed over the range. One should note that this does not imply that $\mathsf{Hash}_\tau(x)$ is a "random oracle" or even a PRF which is a much stronger notion: this would require that for distinct $x_1, \ldots, x_k$, the values $\mathsf{Hash}_\tau(x_1), \ldots, \mathsf{Hash}_\tau(x_k)$ "look" independent and uniformly distributed. The almost uniform distribution of $\mathsf{Hash}_\tau(x)$ does not imply the PRF condition.

For the analysis of most of the schemes, we will not require the almost uniform distribution of $\mathsf{Hash}_\tau(x)$. This will be required at one or two places and will be mentioned explicitly.

2.3 Stream Cipher With IV

Let $\mathsf{SC}_K : \{0,1\}^n \to \{0,1\}^L$ be a stream cipher with IV, i.e., for every choice of K, SC_K maps an IV of length n bits to a string of length L bits. The IV is from an IV-space which is defined to be $\{0,1\}^n$. In other words, initialisation vectors are n-bit strings. The key K is from a suitable (finite) key space. For all practical designs, keys are at least as long as the IVs [2].

The length L is assumed to be long enough for practical sized messages to be encrypted. For example, one can fix L to be equal to 2^{32}. Given an n-bit IV, in practice there is no requirement to generate all L bits of $\mathsf{SC}_K(\mathsf{IV})$. Actual encryption of a message M of ℓ bits using an initialisation vector IV is done by XORing the first ℓ bits of $\mathsf{SC}_K(\mathsf{IV})$ to the message. So, generating ℓ bits are sufficient. By a slight abuse of notation, we will write the encryption of an ℓ-bit message M as $M \oplus \mathsf{SC}_K(\mathsf{IV})$.

In the above, we have taken the output to be a (long) fixed length string. A suitable length prefix of the output is used to encrypt a message. This allows a clean mathematical description of the functionality of a stream cipher with IV. An alternative would be to consider the outputs to be of variable lengths. This causes difficulties in the formulation. Given a particular value of the IV, the length of the output would not be known a priori. It would depend on the length of the message to be encrypted. Our formulation avoids such difficulties.

Consider an algorithm to generate the output of a stream cipher from the inputs K and IV. For most practical stream ciphers [2], such an algorithm goes through two phases. First, there is an

initialisation phase, in which no output is produced. Then, the output bits begin to be produced and this phase is called the keystream generation phase. The speed at which the output bits are produced in the keystream generation phase can be very fast. In comparison, the latency of the initialisation phase can be significant. As a consequence of the latency in the initialisation phase, for most practical stream ciphers, it is advisable to avoid repeated initialisations. We take this aspect into account in our constructions of higher level primitives.

Stream Cipher with IV and PRF: The security assumption on the stream cipher with IV is that of a pseudo-random function. See [6] for a discussion on the justification of this assumption.

A stream cipher with IV can be considered to be a function family $\{\mathsf{SC}_K\}_{K\in\mathcal{K}}$ where $\mathcal{D} = \{0,1\}^n$ (and so, IVs are n-bit strings) and $\mathcal{R} = \{0,1\}^L$. A random function arises from this family through the uniform random choice of a key K. For the PRF-assumption, the requirement is that SC_K is computationally indistinguishable from a function which is chosen uniformly at random from the set of all functions which map $\{0,1\}^n$ to $\{0,1\}^L$. Suppose that the adversary makes a total of q queries. Each query is an IV and is of length n and so the query complexity σ is $q \times n$. So, it is sufficient to count only the number of queries. By $\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t,q)$ we will denote the resource-bounded advantage of an adversary in breaking the PRF-property of SC.

IV usage: Modelling a stream cipher with IV as a PRF restricts the usage of the IV to a *nonce*, i.e., the same value cannot be repeated. More explicitly, separate invocations of a stream cipher have to be done on *distinct* values of the IV.

2.4 Nonce-Based Message Authentication Code

We will consider deterministic MAC algorithms which utilise a nonce. Such a MAC algorithm $\mathsf{MAC}_K(N,M)$ is parameterised by a secret key K and takes as input a nonce N and a message M. The key K is from a finite non-empty set called the key space; the nonce is from the nonce space which is a set of binary strings of some fixed length; the message M is from a message space which consists of all binary strings of some maximum length. In particular, $\mathsf{len}(M)$ can be equal to 0. The upper bound on the length of messages ensures that the message space is finite. The output of the MAC algorithm is a fixed length string $\mathsf{tag} = \mathsf{MAC}_K(N,M)$. The set of all possible tags constitute the tag space.

The key K is secret and is shared between a sender and a receiver. The sender generates tag and sends (N, M, tag) to the receiver; the receiver regenerates the tag from the received nonce and the message and compares to the received tag. If they are equal, the receiver accepts, otherwise, the receiver rejects. Nonce-based MACs have been proposed and Poly1305 [8] is a good example. MAC algorithms are generally defined without nonces and later we will briefly consider this.

Security of nonce-based deterministic MAC schemes is defined using a game in the following manner. An adversary $\mathcal{A}$ is a probabilistic algorithm which has oracle access to the tag generation algorithm $\mathsf{MAC}_K(\cdot,\cdot)$, which is instantiated with a *uniform random key* K. $\mathcal{A}$ adaptively queries the oracle with inputs $(N^{(1)}, M^{(1)}), \ldots, (N^{(q)}, M^{(q)})$ and receives in response the corresponding tags $\mathsf{tag}^{(1)}, \ldots, \mathsf{tag}^{(q)}$. The restriction on $\mathcal{A}$ is that the nonces $N^{(1)}, \ldots, N^{(q)}$ have to be distinct. After the interaction, $\mathcal{A}$ produces a forgery (N, M, tag) such that (N, M) is not equal to $(N^{(i)}, M^{(i)})$ for any $1 \le i \le q$. Note that N may be equal to one of the earlier $N^{(i)}$. Define $\mathsf{succ}(\mathcal{A})$ to be the event that $\mathsf{MAC}_K(N,M) = \mathsf{tag}$, i.e., succ is the event that the forgery provided by the algorithm passes the verification test.

The advantage of $\mathcal{A}$ in breaking the MAC algorithm is defined to be the probability of the event $\mathsf{succ}(\mathcal{A})$.

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{MAC}}(\mathcal{A}) = \Pr[\mathsf{succ}(\mathcal{A})].$$

By $\mathsf{Adv}_{\mathsf{MAC}}(t, q, \sigma)$, we denote the maximum advantage of any adversary running in time t, making q queries (including the forgery attempt) and having query complexity σ. The query complexity counts the number of bits in the nonces and also counts the number of bits in the forgery.

2.5 Authenticated Encryption (With Associated Data)

The goal of such a scheme is to perform both encryption and authentication of a message and also possibly authenticate an additional data or header. There are two deterministic algorithms – the encryption and the decryption algorithms.

The encryption algorithm $\mathsf{AEAD.Encrypt}_K(N, H, M)$ takes as input a nonce N, a (possibly empty) header H and a message M. As before, K is from a suitable key space and N is from the nonce space. The header H can be any binary string of some maximum length and the message can be any *non-empty* binary string of some maximum length. The output of $\mathsf{AEAD.Encrypt}_K(N, H, M)$ is a ciphertext Γ from a suitable ciphertext space. In many AEAD schemes (including the ones we construct), Γ consists of a pair of strings (C, tag) where the length of C is equal to the length of M and tag is considered to be the autheticator. The decryption algorithm $\mathsf{AEAD.Decrypt}_K(N, H, \Gamma)$ takes as input a nonce N, a header H and Γ and either outputs a message M or outputs $\perp$ signifying that the input is rejected. The usual correctness requirement is to be satisfied, namely,

$$\mathsf{AEAD.Decrypt}_K(N, H, \mathsf{AEAD.Encrypt}_K(N, H, M)) = M.$$

We require the message to be a non-empty string. It may be argued that allowing the empty string as a message permits the use of an AEAD scheme as a MAC algorithm. In such an application, the message is fixed to be the empty string and the header can be any non-empty string. Formally speaking, however, the authentication in such a scenario is really of a pair of strings (H, M) where M is the empty string. This is different from authenticating a single string H. In the former case, the input to the MAC algorithm is a 2-component vector, while in the latter case it is a single string. This difference can become important for applications which require to authenticate vectors of strings. As a result, we do not suggest using an AEAD scheme as a MAC algorithm by fixing the message to be the empty string.

There are two security requirements on an AE scheme – confidentiality (or privacy) and authenticity. Below we separately define confidentiality and authenticity.

Privacy of an AEAD scheme. As usual, let an adversary $\mathcal{A}$ be a probabilistic algorithm. $\mathcal{A}$ has oracle access to $\mathsf{AEAD.Encrypt}_K(\cdot, \cdot, \cdot)$ where K is chosen *uniformly at random* and can adaptively query the oracle with inputs $(N^{(1)}, H^{(1)}, M^{(1)}), \ldots, (N^{q)}, H^{(q)}, M^{(q)})$ receiving in response the corresponding outputs $(\Gamma^{(1)}, \ldots, \Gamma^{(q)})$. The restriction on the queries is that the nonces $N^{(1)}, \ldots, N^{(q)}$ must be distinct. Finally, $\mathcal{A}$ outputs a bit. Let $\mathcal{A}^{\mathrm{real}} \Rightarrow 1$ be the event that $\mathcal{A}$ outputs the bit 1 after the interaction.

Now consider the interaction of $\mathcal{A}$ with an oracle which behaves as follows. On input a triple (N, H, M), it returns a uniform random string Γ. The length of Γ is equal to the length of $\mathsf{AEAD.Encrypt}_K(N, H, M)$. This string is independent of all other inputs and responses. Denote by $\mathcal{A}^{\mathrm{rnd}} \Rightarrow 1$ the event that $\mathcal{A}$ outputs the bit 1 after such interaction. The advantage of an adversary $\mathcal{A}$ in breaking the privacy of the AEAD scheme is defined as follows.

$$\mathsf{Adv}^{\text{aead-priv}}_{\mathsf{AEAD}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathrm{real}} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathrm{rnd}} \Rightarrow 1].$$

The maximum of this advantage over all adversaries running in time t, making q queries and having query complexity σ will be denoted by $\mathsf{Adv}^{\text{priv}}_{\mathsf{AEAD}}(t, q, \sigma)$.

Authenticity of an AEAD scheme. As before, the adversary $\mathcal{A}$ is a probabilistic algorithm having oracle access to $\mathsf{AEAD.Encrypt}_K(\cdot, \cdot, \cdot)$ where K is chosen uniformly at random. The adversary also has an implicit access to the decryption oracle as we point out later. The goal of the adversary is to produce a forgery.

$\mathcal{A}$ adaptively queries the oracle with inputs $(N^{(1)}, H^{(1)}, M^{(1)}), \ldots, (N^{q)}, H^{(q)}, M^{(q)})$ and receives in response the corresponding outputs $(\Gamma^{(1)}, \ldots, \Gamma^{(q)})$. The restriction on the queries is that the nonces $N^{(1)}, \ldots, N^{(q)}$ are distinct. At the end of the interaction, $\mathcal{A}$ outputs a tuple (N, H, Γ), where (N, H, Γ) is not equal to $(N^{(i)}, H^{(i)}, \Gamma^{(i)})$ for any $1 \leq i \leq q$. The nonce N in the forgery attempt may be equal to one of the earlier $N^{(i)}$. Define $\mathsf{succ}(\mathcal{A})$ to be the event that $\mathsf{AEAD.Decrypt}_K(N, H, \Gamma)$ does not return $\perp$. Note that the definition of $\mathsf{succ}(\mathcal{A})$ requires an implicit access to the decryption oracle of the AEAD scheme. The advantage of an adversary $\mathcal{A}$ in breaking the authenticity of the AEAD scheme is defined as follows.

$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD}}(\mathcal{A}) = \Pr[\mathsf{succ}(\mathcal{A})].$$

The maximum of this advantage over all adversaries running in time t, making q queries and having query complexity σ will be denoted by $\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD}}(t, q, \sigma)$. The number of queries q includes the forgery attempt and the query complexity counts the bits in the nonces.

AE Schemes. These are defined in a manner similar to that of AEAD schemes. The only difference is that there is no header or associated data in an AE scheme. Definitions of privacy and authenticity of AE schemes are obtained from that of AEAD schemes by dropping the header. The superscripts ae-priv and ae-auth will be used to denote privacy and authenticity for AE schemes.

2.6 Deterministic Authenticated Encryption (With Associated Data)

This is an authenticated encryption (with associated data) scheme which does not use a nonce. Syntactically, such a scheme is almost the same as that of an AEAD scheme.

It consists of two algorithms $\mathsf{DAEAD.Encrypt}_K(H, M)$ and $\mathsf{DAEAD.Decrypt}_K(H, \Gamma)$, where K is from a suitable key space, H is the header, M is the message and Γ is the ciphertext. Depending on the application, the header H can be a single binary string; or, a vector consisting of a fixed number of binary strings; or, a vector of binary strings where the number of such strings can vary.

The encryption algorithm produces Γ as output and the decryption algorithm either rejects its input (by producing the symbol $\perp$) or produces a message. Soundness is ensured by the following condition.

$$\mathsf{DAE.Decrypt}_K(H, \mathsf{DAE.Encrypt}_K(H, M)) = M.$$

Security of DAEAD schemes are defined in much the same way as that for AE schemes. There are two parts – privacy and authenticity. (See [33] for an approach where the two parts can be merged into an equivalent single security model.) Since the scheme is deterministic, the adversary is not allowed to repeat a query to the encryption oracle. For authenticity, a forgery attempt is a triplet (H, C, tag) where (C, tag) is not equal to the output of any possible previous query (H, M) to the encryption oracle. As in the case of AEAD, we denote by $\mathsf{Adv}^{\text{daead-priv}}_{\mathsf{D}}(t, q, \sigma)$ (resp. $\mathsf{Adv}^{\text{daead-auth}}_{\mathsf{D}}(t, q, \sigma)$) the maximum advantage of any adversary in breaking the privacy (resp. the authenticity) of a DAEAD scheme D, where the maximum is over all adversaries running in time

t, making q queries and having query complexity σ (i.e., sending a total of σ bits in all its queries). For authenticity, the number of queries includes the forgery attempt.

A DAE scheme is a special case of a DAEAD scheme where there is no header. Conceptually, a DAE scheme is not much different from an AE scheme. (Similarly, for DAEAD and AEAD schemes.) This can be seen by considering the nonce-message pair of an AE scheme to be a message of a DAE scheme. In the security model of an AE scheme, the nonce cannot be repeated, while in a DAE scheme, the message cannot be repeated. So, considering the nonce to be a part of the message ensures that messages are not repeated.

This, however, causes a difference in how the two primitives are constructed. While it is true that messages in a DAE scheme are distinct, it is not necessarily true that there is a specific location of n bits which have distinct values for different messages. Rather the entire message needs to be considered as a nonce. This makes the construction of DAE schemes significantly more inefficient compared to an AE scheme.

In broad terms, a DAE scheme can always be used as an AE scheme: simply consider the pair (N, M) as the message in the DAE scheme. But, the converse is not true, i.e., an AE scheme cannot be used as a DAE scheme.

Note: For AE(AD) and DAE(AD), two security conditions are defined – privacy and authenticity. These are defined against adversaries which can make oracle queries. As per the convention we have defined and will be following, an adversary against privacy makes q queries whereas an adversary against authenticity makes $q - 1$ oracle queries and provides a forgery attempt.

3 Vector-Input Hash Functions

Rogaway and Shrimpton [33] consider the problem of constructing a PRF which can handle a vector of strings as input. They propose an iterative method which repeatedly applies a single-input PRF successively to the different components of the vector. It is mentioned in [33] that this procedure is more efficient than using a padding rule to first convert the vector of strings to a single string and then applying a single-input PRF to this single string. In the context that [33] considers, this is perhaps correct. The context is to instantiate the single-input PRF by a block cipher so that the construction essentially becomes a mode of operation of a block cipher. As such, the parameter of interest is the number of block cipher invocations to process the vector.

In the context of a stream cipher with IV, however, the parameter of interest changes. Considered as a PRF, the input is the IV and the output is the (suitably truncated) keystream. Thus, each application of the PRF involves an initialisation of the stream cipher. As discussed earlier, for most practical stream ciphers [2] the latency of the initialisation phase is usually significant compared to the time per byte of the key generation phase. So, for using a stream cipher with IV, it is of interest to reduce the number of times the stream cipher is applied to different IVs. Using the approach of [33] to handle a vector of strings will mean that the stream cipher is invoked with different IVs and hence will be repeatedly initialised. Instead, it is better to encode the vector of strings into a single string and apply a hash function to the single string. The output of the hash function is then used as the IV to the stream cipher. In this approach, the stream cipher is initialised only once.

Iwata and Yasuda [20] in a work subsequent to [33] describe a vector-input hash function with low differential probabilities. The construction is based on polynomial hashing and briefly is the following. Suppose the field $GF(2^n)$ is represented by the polynomial $\rho(x)$ which is irreducible and of degree n over $GF(2)$. Elements of $GF(2^n)$ are considered to be binary polynomials of degree less than n with addition and multiplication carried out modulo $\rho(x)$.

Let τ be the n-bit hash key which is considered to be an element of $GF(2^n)$. Let the vector of inputs to the hash function be $(X_1, \ldots, X_\ell)$. For each i, if the length of X_i is not a multiple of n, then it is padded with 10^* to the next multiple of n; set a constant (over $GF(2^n)$) c_i to be x if X_i required padding; otherwise set c_i to be 1. Suppose the blocks of the padded X_i are $X_{i,0}, \ldots, X_{i,m_i-1}$ and define $Z_i = \tau^{m_i} \oplus \tau^{m_i-1} X_{i,0} \oplus \cdots \oplus X_{i,m_i-1}$ which is the usual Horner's rule based polynomial hash. Each Z_i is an element of $GF(2^n)$ and so can be represented by a polynomial of degree at most $n-1$ over $GF(2)$. We write $Z_i(x)$ to emphasize this representation. Similarly, let $\tau(x)$ be the polynomial representing the secret key τ. The final digest of the hash function for the input $(X_1, \ldots, X_\ell)$ is the following:

$$\tau(x)Z_1(x)c_1 \oplus \tau^2(x)Z_2(x)c_2 \oplus \cdots \oplus \tau^\ell(x)Z_\ell(x)c_\ell \pmod{\rho(x)}.$$

Multiplication by x can be carried out efficiently and require very little time. Hence, the time required for multiplications by the constants c_is can be ignored. The main cost is the number of multiplications over $GF(2^n)$. This number is $\ell + \sum_{i=1} \ell m_i$. Essentially, the construction first pads the strings; hashes them using polynomial hashing; and then hashes the digests (after tweaking them with the constants) again using polynomial hashing.

Using the Iwata-Yasuda hash function with a stream cipher will avoid the problem of repeated initialisations. However, it is a specific construction based on polynomial hashing. Our goal, on the other hand, is more general. We wish to obtain efficient methods for converting a single-input hash function to a vector-input hash function which will work for *all* possible single-input hash functions. This involves several methods for encoding a vector of binary strings into a single binary string and also suitably composing such encodings with a single-input hash function.

3.1 Encoding Methods

We describe several methods to map a vector of strings to a single string. The parameters of the encoding method are $\beta \geq 1$ and $\mathbf{w} = (w_1, \ldots, w_\kappa)$ where each $w_i \geq 1$. Formally, the encoding method $\mathcal{E}$ with parameters $\mathbf{w}$ and β is denoted as $\mathcal{E}[\mathbf{w}, \beta]$. Given a vector of strings $(A_1, \ldots, A_\kappa)$ with the length of A_i being *at most* $2^{w_i} - 1$, the output of the encoding is the following:

$$\mathcal{E}_1[\mathbf{w}, \beta](A_1, \ldots, A_\kappa) = A_1 || \cdots || A_\kappa || \mathsf{bin}_{w_1}(A_1) || \cdots || \mathsf{bin}_{w_{\kappa-1}}(A_{\kappa-1}) || 10^k \tag{2}$$

where k is the minimum non-negative integer such that the total length of the right hand side is a multiple of β. We allow the maximum lengths of the A_is to be different. In AEAD applications, $\kappa = 2$ and A_1 is the header while A_2 is the message. The maximum length of the header may be significantly smaller than the maximum length of the message. By allowing $w_1 < w_2$, the output of the above encoding leads to a shorter string.

Instead of appending the lengths at the end of the message, it is possible to append the length of an input component immediately after the component appears as follows.

$$\mathcal{E}_2[\mathbf{w}, \beta](A_1, \ldots, A_\kappa) = A_1 || \mathsf{bin}_{w_1}(\mathsf{len}(A_1)) || \cdots || A_\kappa || \mathsf{bin}_{w_\kappa}(\mathsf{len}(A_\kappa)) || 10^k \tag{3}$$

where as before k is the least non-negative integer such that the length of the right hand side is a multiple of β.

An advantage of doing this is that one does not need to carry forward the encodings of the different lengths to be appended after all the inputs have been read. On the other hand, this

requires the length of A_κ also to be appended and makes the resulting string longer. Note that the simpler encoding scheme

$$A_1||\mathsf{bin}_{w_1}(\mathsf{len}(A_1))||\cdots||A_{\kappa-1}||\mathsf{bin}_{w_{\kappa-1}}(\mathsf{len}(A_{\kappa-1}))||A_\kappa||10^k$$

will not work. To see this suppose $\kappa = 2$, $\beta = 4$ and $w_1 = w_2 = 2$, i.e., the block length is 4 and strings are of maximum length 3. Let $(A_1, A_2) = (1, 01)$ and $(A'_1, A'_2) = (10, 1)$. Then the encoding of (A_1, A_2) is $1||01||01||100$ and the encoding of (A'_1, A'_2) is $10||10||1||100$ and so the two encodings are equal.

We introduce notation to denote the restricted encodings where $w_1 = w_2 = \cdots = w_\kappa = w$. For $\kappa \geq 1$ and binary strings $A_1, \ldots, A_\kappa$ each of maximum length $2^w - 1$, the output of $\mathcal{E}_3[w, \beta]$ is defined to be the following.

$$\mathcal{E}_3[w, \beta](A_1, \ldots, A_\kappa) = A_1||\mathsf{bin}_w(\mathsf{len}(A_1))||\cdots||A_\kappa||\mathsf{bin}_w(\mathsf{len}(A_\kappa))||10^k \tag{4}$$

where as before k is the least non-negative integer such that the length of the right hand side is a multiple of β.

These encodings will be used to construct vector-input hash functions. In the following, we state and prove certain properties of the encodings which will later be used to argue about the differential probabilities of the constructed hash functions.

Proposition 1. *Let $\kappa \geq 1$, $(A_1, \ldots, A_\kappa) \neq (A'_1, \ldots, A'_\kappa)$ and set*

$$C = \mathcal{E}_1[\mathbf{w}, \beta](A_1, \ldots, A_\kappa),\ C' = \mathcal{E}_1[\mathbf{w}, \beta](A'_1, \ldots, A'_\kappa).$$

Assume that $\mathsf{len}(C) = \ell \geq \ell' = \mathsf{len}(C')$. Then

$$C \neq C'||0^{\ell-\ell'}. \tag{5}$$

Further, (5) holds when $\mathcal{E}_1$ is replaced by $\mathcal{E}_2$.

Note that by definition of $\mathcal{E}_1$ (and also $\mathcal{E}_2$) both ℓ and ℓ' are multiples of β and hence, so is the difference $\ell - \ell'$.

Proof. By definition $C = A_1||\cdots||A_\kappa||\mathsf{bin}_{w_1}(A_1)||\cdots||\mathsf{bin}_{w_{\kappa-1}}(A_{\kappa-1})||10^k$ and let

$$D = A_1||\cdots||A_\kappa||\mathsf{bin}_{w_1}(A_1)||\cdots||\mathsf{bin}_{w_{\kappa-1}}(A_{\kappa-1}).$$

Similarly, $C' = A'_1||\cdots||A'_\kappa||\mathsf{bin}_{w_1}(A'_1)||\cdots||\mathsf{bin}_{w_{\kappa-1}}(A'_{\kappa-1})||10^{k'}$ and let

$$D' = A'_1||\cdots||A'_\kappa||\mathsf{bin}_{w_1}(A'_1)||\cdots||\mathsf{bin}_{w_{\kappa-1}}(A'_{\kappa-1}).$$

Let $m = \mathsf{len}(D)$ and $m' = \mathsf{len}(D')$.

If $m > m'$ the $(m+1)$st bit of C is 1 while the $(m+1)$st bit of $C'||0^{\ell-\ell'}$ is 0. So, in this case, $C \neq C'||0^{\ell-\ell'}$.

Suppose now that $m = m'$ and then $k = k'$ necessarily follows and as a result $\ell = \ell'$. The condition $m = m'$ means that D and D' are equal length strings and the condition $\ell = \ell'$ means that C and C' are equal length strings. These two conditions force $k = k'$. Consider the last $w_1 + \cdots + w_{\kappa-1}$ bits of D and D'. If these are unequal, then $D \neq D'$ and so $C \neq C'$. On the other hand, if these last bits are equal, then the lengths of A_i and A'_i are equal for $i = 1, \ldots, \kappa - 1$. Since the lengths of D and D' are equal, this forces the lengths of A_κ and A'_κ to be equal. So, the lengths of

A_i and A'_i are equal for $i = 1, \ldots, \kappa$. From this and the hypothesis that $(A_1, \ldots, A_\kappa) \neq (A'_1, \ldots, A'_\kappa)$, it follows that $D \neq D'$ leading to $C \neq C'$. Thus, in all cases, $C \neq C'||0^{\ell-\ell'}$.

The proof of the statement regarding $\mathcal{E}_2$ is a little different. The case when $m > m'$ is the same. For the case $m = m'$, as before, the lengths of D and D' are equal and so are the lengths of C and C'. Again, as before, $k = k'$. Strip off the last k bits from both C and C' leaving D and D' respectively.

Consider the last w_κ bits of both D and D'. If these are unequal, then clearly D and D' are unequal. On the other hand, if these are equal, then the lengths of A_κ and A'_κ are equal. If these two strings are unequal, then again D and D' are unequal. Otherwise, strip off $\mathsf{len}(A_\kappa) + w_\kappa$ bits from D and D' and apply the same reasoning to the shorter strings. Continuing this by backward induction, we will either get D and D' to be unequal, or we will get $A_i = A'_i$ for $i = 1, \ldots, \kappa$. Since the last condition is ruled out by hypothesis, it follows that $D \neq D'$ and so $C \neq C'$. □

Proposition 2. *Let $\kappa \geq 1$, $(A_1, \ldots, A_\kappa) \neq (A'_1, \ldots, A'_\kappa)$ and set*

$$C = \mathcal{E}_2[\mathbf{w}, \beta](A_1, \ldots, A_\kappa),\ C' = \mathcal{E}_2[\mathbf{w}, \beta](A'_1, \ldots, A'_\kappa).$$

Assume that $\mathsf{len}(C) = \ell \geq \ell' = \mathsf{len}(C')$ and let E be the suffix of C of length ℓ'. Then $E \neq C'$.

Proof. The argument is similar to the argument given for $\mathcal{E}_2$ in the proof of Proposition 1. The argument proceeds backwards from the ends of the strings E and C'. If $k \neq k'$, then clearly the last ones in E and C' are at different positions and so the strings are unequal. So, assume that $k = k'$ and then strip off $k + 1$ bits from the ends of both E and C'. Consider now the last w_κ bits of the two shorter strings. If these are not equal, then E and C' are unequal. If they are equal, then the lengths of A_κ and A'_κ are equal. If these two strings are unequal, then again E and C' are unequal. Otherwise strip off further $\mathsf{len}(A_\kappa) + w_\kappa$ bits and apply the same argument. Continuing, we will either get E and C' to be unequal, or, $A_i = A'_i$ for $i = 1, \ldots, \kappa$. Since the last condition is ruled out by hypothesis, we must have E and C' to be unequal. □

Proposition 3. *Let $\kappa, \kappa' \geq 1$, $(A_1, \ldots, A_\kappa) \neq (A'_1, \ldots, A'_{\kappa'})$ and set*

$$C = \mathcal{E}_3[w, \beta](A_1, \ldots, A_\kappa),\ C' = \mathcal{E}_3[w, \beta](A'_1, \ldots, A'_{\kappa'}).$$

Assume that $\mathsf{len}(C) = \ell \geq \ell' = \mathsf{len}(C')$. Then the following holds.

1. *$C \neq C'||0^{\ell-\ell'}$.*
2. *Let E be the suffix of C of length ℓ'. Then $E \neq C'$.*

Proof. Unlike the previous proofs, in this case, the number of components in the two input vectors need not be equal. The argument, though, is similar to the previous proofs.

Consider the first point. If $\ell > \ell'$, then $C'||0^{\ell-\ell'}$ has a 0 at the position where C has the last 1 and so the result follows. On the other hand, if $\ell = \ell'$, then we proceed from the ends of the strings C and C' and argue backwards as in the proof of Proposition 1. This will either result in $C \neq C'$ or, we will end up with $\kappa = \kappa'$ and $A_i = A'_i$ for $i = 1, \ldots, \kappa$. Since the last condition is ruled out by the hypothesis, we must certainly have $C \neq C'$.

For the second point, we proceed as in the proof of Proposition 2. Suppose k (resp. k') zeros have been padded to obtain C (resp. C'). If $k \neq k'$, then the last ones in E and C' are in different positions and so $E \neq C'$. Assuming $k = k'$, strip off the last $k + 1$ bits from the ends of both E and C'. The next w bits of both E and C' encode the lengths of A_κ and $A'_{\kappa'}$. (Note that here we make use of the fact that the lengths of both A_κ and $A'_{\kappa'}$ are encoded as w-bit strings.) If these

encodings are not equal, then $E \neq C'$. If they are equal, then we compare A_κ and $A'_{\kappa'}$; if these are not equal, then again $E \neq C'$; if they are equal, then we strip off $A_\kappa||\mathsf{bin}_w(\mathsf{len}(A_\kappa))$ from the ends of both E and C' and proceed by backward induction. This will either show that $E \neq C'$ or, show that $\kappa = \kappa'$ and $A_i = A'_i$ for $i = 1, \ldots, \kappa$. Again, since the last condition is ruled out by hypothesis, we get the desired result. □

3.2 Vector Input Hash Functions: AXU for Equal Length Vectors

We describe constructions of hash functions which take as input a vector of strings such that the differential probabilities are provably low for two vectors of the *same* length, i.e., for two vectors having the *same* number of components.

Let h_τ be a single-input hash function which produces an n-bit digest and takes as inputs strings whose lengths are positive multiples of some block length β. The block length need not be equal to the digest length n.

Given a single-input hash function h_τ and an encoding method $\mathcal{E}$, we define a hash function parameterised by h_τ and $\mathcal{E}$ as follows.

$$\mathsf{Hash1}_\tau[h, \mathcal{E}](A_1, \ldots, A_\kappa) = h_\tau(\mathcal{E}[\mathbf{w}, \beta](A_1, \ldots, A_\kappa)). \tag{6}$$

In other words, the vector $(A_1, \ldots, A_\kappa)$ is first mapped to the string $C = \mathcal{E}[\kappa, \mathbf{w}, \beta](A_1, \ldots, A_\kappa)$ and then h_τ is applied to C.

Theorem 1. *Suppose the differential probabilities of* $\{h_\tau\}_\tau$ *for* equal length *strings are at most* ε. *Further, suppose that for every* τ, *any string* C *and all* $i \geq 0$,

$$h_\tau(C) = h_\tau(C||0^{\beta i}). \tag{7}$$

Let $(A_1, \ldots, A_\kappa) \neq (A'_1, \ldots, A'_\kappa)$. *Then for every* $\alpha \in GF(2^n)$

$$\Pr[\mathsf{Hash1}_\tau[h, \mathcal{E}_1](A_1, \ldots, A_\kappa) \oplus \mathsf{Hash1}_\tau[h, \mathcal{E}_1](A'_1, \ldots, A'_\kappa) = \alpha] \leq \varepsilon.$$

Here the probability is taken over the uniform random choice of τ. *Further, the result holds if* $\mathcal{E}_1$ *is replaced by* $\mathcal{E}_2$.

The property given by (7) will be referred to as the *null extension* property of h_τ.

Proof. Let $C = \mathcal{E}_1[\mathbf{w}, \beta](A_1, \ldots, A_\kappa)$ and $C' = \mathcal{E}_1[\mathbf{w}, \beta](A'_1, \ldots, A'_\kappa)$. Assume without loss of generality that $\mathsf{len}(C) = \ell \geq \ell' = \mathsf{len}(C')$. From Proposition 1 we have $C \neq C'||0^{\ell - \ell'}$.

By the null extension property, $h_\tau(C') = h_\tau(C'||0^{\ell-\ell'})$. So,

$$\begin{aligned}
&\Pr[h^{(\kappa)}_\tau(A_1, \ldots, A_\kappa) \oplus h^{(\kappa)}_\tau(A'_1, \ldots, A'_\kappa) = \alpha] \\
&= \Pr[h_\tau(\mathcal{E}[\mathbf{w}, \beta](A_1, \ldots, A_\kappa)) \oplus h_\tau(\mathcal{E}[\mathbf{w}, \beta](A'_1, \ldots, A'_\kappa)) = \alpha] \\
&= \Pr[h_\tau(C) \oplus h_\tau(C') = \alpha] \\
&= \Pr[h_\tau(C) \oplus h_\tau(C'||0^{\ell-\ell'}) = \alpha] \\
&\leq \varepsilon.
\end{aligned}$$

The last inequality follows from the differential property of $\{h_\tau\}_\tau$ on equal length strings.

The crucial property of $\mathcal{E}_1$ used in the above argument is that $C \neq C'||0^{\ell-\ell'}$. From Proposition 1, this property also holds for the encoding method $\mathcal{E}_2$ and so the statement about $\mathcal{E}_2$ also holds. □

Several well known hash functions satisfy the conditions of Theorem 1. We discuss these below.

Multi-Linear Hash Function [17, 43]: Let $\mathbb{F}$ be a finite field such that the elements of $\mathbb{F}$ can be encoded as β-bit strings. An input to h_τ is a sequence $M_0, M_1, \ldots, M_{t-1}$ of β-bit strings considered to be elements of $\mathbb{F}$. The key τ is also a sequence of $K_0, K_1, \ldots$ of elements from $\mathbb{F}$. The output of h_τ is defined to be

$$Y = K_0M_0 + K_1M_1 + \cdots + K_{t-1}M_{t-1}.$$

It is easy to show that for equal length sequences, the differential probabilities of $\{h_\tau\}_\tau$ are bounded above by $1/\#\mathbb{F}$. The portion of the key that is used to hash an input is as long as the input itself. It is easy to see appending all-zero β-bit blocks does not change the value of the digest and so the null extension property holds for h_τ defined as above. In this construction, the digest is an element of $\mathbb{F}$ and so the digest can be represented using β bits.

A variant of the multi-linear hash function is the so-called Toeplitz construction. As above, let the input be a sequence $M_0, M_1, \ldots, M_{t-1}$ of elements of $\mathbb{F}$. The key as before is a sequence $K_0, K_1, \ldots$ of elements from $\mathbb{F}$ where b is a parameter to the construction. The output is defined to be $Y_0||Y_1||\cdots||Y_{b-1}$ where Y_i are β-bit strings representing elements of $\mathbb{F}$ and are computed as follows.

$$\begin{aligned}
Y_0 &= K_0M_0 + K_1M_1 + \cdots + K_{t-1}M_{t-1} \\
Y_1 &= K_1M_0 + K_2M_1 + \cdots + K_tM_{t-1} \\
&\cdots \\
Y_{b-1} &= K_{b-1}M_0 + K_bM_1 + \cdots + K_{t+b-1}M_{t-1}.
\end{aligned}$$

Here the size n of the digest is $n = b\beta$ whereas the input blocks are each β-bit strings. To hash a t-block input, $t + b$ key blocks are required. It is known that the differential probabilities of such a hash function for equal length sequences is $1/(b \times \#\mathbb{F})$. Clearly, this construction also satisfies the null extension property.

Pseudo-Dot Product [44]: A method of evaluating inner products was described by Winograd [44] and in [10] it was pointed out that this leads to a fast hash function. Let $\mathbb{F}$ be a finite field such that its elements can be encoded as γ-bit strings and set $\beta = 2\gamma$. The input to the hash function is a sequence of $M_0, \ldots, M_{2t-1}$ of γ-bit strings considered to be elements of $\mathbb{F}$. (Note that this sequence can be also be seen as a sequence of length t consisting of β-bit strings.) The key $K_0, K_1, \ldots$ is also a sequence of elements from $\mathbb{F}$. The output is a γ-bit string and is defined to be the following.

$$(M_0 + K_0)(M_1 + K_1) + (M_2 + K_2)(M_3 + K_3) + \cdots + (M_{2t-2} + K_{2t-2})(M_{2t-1} + K_{2t-1}).$$

Appending all-zero β-bit blocks to the input does not change the digest. So, this hash function satisfies the null extension property. The differential probabilities are bounded above by $1/\#\mathbb{F}$.

As in the case of multi-linear hash function, a Toeplitz version of the pseudo-dot product can be defined hashing the input b times using shifted versions of the key. The idea is similar except that the successive shifts of the key is by two blocks, i.e., β bits. The digest is of size $b\gamma$ and the differential probabilities are bounded above by $1/(b \times \mathbb{F})$. Again, it is easy to check that the null extension property holds for the Toeplitz version.

NH *and* NH^T *[11]:* A variant of the pseudo-dot product construction has been proposed in [11] and has been called the NH hash function. Instead of a finite field, the construction works with integer arithmetic. The input $M_0, M_1, \ldots, M_{2t-1}$ consists of γ-bit strings and set $\beta = 2\gamma$. The

key $K_0, K_1, \ldots$ also consists of γ-bit strings. The output of the hash function is defined to be the following quantity:

$$(K_0 +_\gamma M_0)(K_1 +_\gamma M_1) + \cdots + (K_{2t-2} +_\gamma M_{2t-2})(K_{2t-1} +_\gamma M_{2t-1}) \pmod{2^\beta}.$$

Here $+_\gamma$ denotes addition modulo 2^γ and the final sum is evaluated modulo 2^β. The digest is a β-bit string and the collision probabilities are bounded above by $1/2^\gamma$. (Differential probabilities have been analysed in [23].) Appending all-zero β-bit blocks to the input clearly does not change the digest and so the hash function satisfies the null extension property. The Toeplitz variant of NH has also been defined in [11]. This has been denoted NH^T and a bound on the collision probability of NH^T has been obtained. Again, NH^T satisfies the null extension property.

Polynomial Hashing: As before, let $\mathbb{F}$ be a finite field and let β be such that elements of $\mathbb{F}$ can be encoded as β-bit strings. The input $M_0, \ldots, M_{t-1}$ to the hash function consist of elements of $\mathbb{F}$. The key τ is a single element of $\mathbb{F}$. The output is defined to be the following:

$$M_0\tau + M_1\tau^2 + \cdots + M_{t-1}\tau^t.$$

The output is an element of $\mathbb{F}$ (and so a β-bit string). The differential probabilities are bounded above by $t/\#\mathbb{F}$. Appending all-zero β-bit blocks to the input does not change the digest and so the function satisfies the null extension property.

There is one problem, however, with this construction. The usual efficient way of evaluating a polynomial is Horner's rule. If the input blocks $M_0, M_1, \ldots$ are provided as input in order, then applying Horner's rule results in the output to be defined as

$$\mathsf{Horner}_\tau(M_0, \ldots, M_{t-1}) = M_0\tau^t + M_1\tau^{t-1} + \cdots + M_{t-1}\tau. \tag{8}$$

While the differential probability remains unchanged, this function no longer satisfies the null extension property. So, Theorem 1 cannot be applied to polynomial hashing defined in this form. We prove a separate result to tackle this case.

Theorem 2. *Let*
$(A_1, \ldots, A_\kappa) \neq (A'_1, \ldots, A'_\kappa)$*;*
$C = \mathcal{E}_2[\mathbf{w}, \beta](A_1, \ldots, A_\kappa)$ *and* $C' = \mathcal{E}_2[\mathbf{w}, \beta](A'_1, \ldots, A'_\kappa)$*;*
$\mathsf{len}(C) = \ell \geq \ell' = \mathsf{len}(C')$.
Then for every $\alpha \in GF(2^n)$

$$\Pr[\mathsf{Hash1}_\tau[\mathsf{Horner}, \mathcal{E}_2](A_1, \ldots, A_\kappa) \oplus \mathsf{Hash1}_\tau[\mathsf{Horner}, \mathcal{E}_2](A'_1, \ldots, A'_\kappa) = \alpha] \leq t/\#\mathbb{F}.$$

Here the probability is taken over the uniform random choice of τ *and* t *is the number of* β*-bit blocks in* C*.*

Note that the encoding method $\mathcal{E}_2$ is used in this case.

Proof. Let $C = \mathcal{E}_2[\mathbf{w}, \beta](A_1, \ldots, A_\kappa)$ and $C' = \mathcal{E}_2[\mathbf{w}, \beta](A'_1, \ldots, A'_\kappa)$. Without loss of generality assume that $\mathsf{len}(C) = \ell \geq \ell' = \mathsf{len}(C')$. Let E be the suffix of C of length ℓ'. From Proposition 2, we have $E \neq C'$.

Both ℓ and ℓ' are multiples of β and we can parse C and C' into β-bit blocks. Let the blocks of C be $M_0, \ldots, M_{t-1}$ and those of C' be $M'_0, \ldots, M'_{t'-1}$. Then E consists of the blocks $M_{t-t'}, \ldots, M_{t-1}$.

The condition $E \neq C'$ ensures that there is an i in $\{0, \ldots, t'-1\}$ such that $M_{t-t'+i} \neq M'_i$. As a result,

$$\mathsf{Hash1}_\tau[\mathsf{Horner}, \mathcal{E}_2](A_1, \ldots, A_\kappa) \oplus \mathsf{Hash1}_\tau[\mathsf{Horner}, \mathcal{E}_2](A'_1, \ldots, A'_\kappa)$$

is a non-zero polynomial over $\mathbb{F}$ of degree at most t and from this the result follows. □

GCM [24] uses a hash function called GHASH which uses polynomial based hashing using Horner's rule. The construction is essentially a double-input hash function, though, this is not explicitly mentioned. A fast polynomial based hash function is Poly-1305 [8]. The description of Poly-1305 is at the byte level. A message M consists of a sequence of bytes. These are formatted into segments where each segment consists of 16 bytes and the last segment has possibly less than 16 bytes. A one is appended to each segment; for the last segment, after appending the one, further zeros are appended (if required) so that the segment length comes to 129 bits. As a result of this padding, each segment length becomes 129 bits. Denote by $\mathcal{E}$-Poly1305(M) the output of this padding scheme. It has been shown in [8], that for any integer u, if $\mathcal{E}$-Poly1305$(M) -$ $\mathcal{E}$-Poly1305$(M') - u \equiv 0 \bmod 2^{130} - 5$, then $M = M'$.

The digest is obtained by using Horner's rule to evaluate a polynomial at a point which is equal to the key of the hash function. This polynomial is defined by the segments which form the coefficients of the polynomial. Denote the digest as $\mathsf{Horner}_\tau(\mathcal{E}\text{-Poly1305}(M))$, where τ is the key.

Poly1305 can be extended to handle vector inputs in the following manner. Fix $\kappa \geq 1$ and let $w_1 = \cdots = w_\kappa = 32$, i.e., each component of the input vector has maximum length $2^{32} - 1$. Since Poly1305 operates at byte level, we set $\beta = 8$. Then the vector extension of Poly1305 is defined as follows.

$$(A_1, \ldots, A_\kappa) \xmapsto{\tau} \mathsf{Horner}_\tau(\mathcal{E}\text{-Poly1305}(\mathcal{E}_2[\kappa, (32, \ldots, 32), 8](A_1, \ldots, A_\kappa))).$$

Following Theorem 2, it is possible to show that the differential probabilites of this map are the same as those of Poly1305. The basic idea is modular, i.e., the vector of strings $(A_1, \ldots, A_2)$ is encoded using $\mathcal{A}_2$ and then simply Poly1305 is applied to the resulting string. In other words, one only needs to apply a simple wrapper layer on top of Poly1305.

A family of fast (single-input) hash functions have been proposed by Bernstein [10] based on an earlier work by Rabin and Winograd [27] and the family has been called the BRW functions in [37]. This family does not satisfy the null-extension property and neither can the method of Theorem 2 be applied to it. So, the BRW hash function cannot be extended to handle vector inputs using the methods described so far. More generally, we would like to obtain an extension technique which does not depend on special properties of the underlying hash function.

Let $\{\psi_\rho\}_\rho$ be a family of functions where for each ρ, ψ_ρ maps n bits to n bits and satisfies the following differential property: For $\gamma \neq \gamma'$ and for any n-bit string α, $\Pr[\psi_\rho(\gamma) \oplus \psi_\rho(\gamma') = \alpha] = 1/2^n$. Such functions ψ can be constructed using the so-called powering-up method [29] and from (word-oriented) linear feedback shift registers [36].

The construction of the general vector-input hash function is based on ψ_ρ, the encoding method $\mathcal{E}_1$ and a single-input hash function h_τ. The key to the extended hash function consists of the pair (ρ, τ).

$$\mathsf{Hash2}_{\tau,\rho}[h, \psi, \mathcal{E}_1](A_1, \ldots, A_\kappa) = h_\tau(\mathcal{E}_1[\kappa, \mathbf{w}, \beta](A_1, \ldots, A_\kappa)) \oplus \psi_\rho(\mathsf{bin}_n(r)) \tag{9}$$

where r is the number of β-bit blocks in $\mathcal{E}_1[\kappa, \mathbf{w}, \beta](A_1, \ldots, A_\kappa)$.

Theorem 3. *Suppose the differential probabilities of* $\{h_\tau\}_\tau$ *for* equal length *strings are at most* ε. *Let* $(A_1, \ldots, A_\kappa) \neq (A'_1, \ldots, A'_\kappa)$. *Then for every* $\alpha \in GF(2^n)$

$$\Pr[\mathsf{Hash2}_{\tau,\rho}[h, \psi, \mathcal{E}_1](A_1, \ldots, A_\kappa) \oplus \mathsf{Hash2}_{\tau,\rho}[h, \psi, \mathcal{E}_1](A'_1, \ldots, A'_\kappa) = \alpha] \leq \varepsilon.$$

Here the probability is taken over the uniform random choice of (τ, ρ).

Proof. Let $C = \mathsf{Hash2}_{\tau,\rho}[h, \psi, \mathcal{E}_1](A_1, \ldots, A_\kappa)$, $C' = \mathsf{Hash2}_{\tau,\rho}[h, \psi, \mathcal{E}_1](A'_1, \ldots, A'_\kappa)$ and the number of β-bit blocks in C and C' be r and r' respectively. First suppose that $r \neq r'$. Then

$$\begin{aligned}
&\Pr[\mathsf{Hash2}_{\tau,\rho}[h, \psi, \mathcal{E}_1](A_1, \ldots, A_\kappa) \oplus \mathsf{Hash2}_{\tau,\rho}[h, \psi, \mathcal{E}_1](A'_1, \ldots, A'_\kappa) = \alpha] \\
&= \Pr[h_\tau(C) \oplus \psi_\rho(\mathsf{bin}_n(r)) \oplus h_\tau(C') \oplus \psi_\rho(\mathsf{bin}_n(r')) = \alpha] \\
&= \Pr[\psi_\rho(\mathsf{bin}_n(r)) \oplus \psi_\rho(\mathsf{bin}_n(r')) = \alpha \oplus h_\tau(C) \oplus h_\tau(C')] \\
&\leq \frac{1}{2^n}.
\end{aligned}$$

The last inequality follows from the differential property of ψ.

Now suppose that $r = r'$ and so the lengths of C and C' are equal. Using Proposition 1, we have $C \neq C'$. Then

$$\begin{aligned}
&\Pr[\mathsf{Hash2}_{\tau,\rho}[h, \psi, \mathcal{E}_1](A_1, \ldots, A_\kappa) \oplus \mathsf{Hash2}_{\tau,\rho}[h, \psi, \mathcal{E}_1](A'_1, \ldots, A'_\kappa) = \alpha] \\
&= \Pr[h_\tau(C) \oplus \psi_\rho(\mathsf{bin}_n(r)) \oplus h_\tau(C') \oplus \psi_\rho(\mathsf{bin}_n(r')) = \alpha] \\
&= \Pr[h_\tau(C) \oplus h_\tau(C') = \alpha] \\
&\leq \varepsilon.
\end{aligned}$$

The last inequality follows from the differential property of h_τ. Since $1/2^n \leq \varepsilon$, the result follows. □

Notes:

1. For both Hash1 and Hash2, the value of κ should be at least 1. Apart from this there are no other restrictions on κ. So, these constructions can be applied on vectors having different number of components. The guarantee on differential probabilities, however, hold only for two vectors having the same number of components.
2. For AEAD applications, we will require to hash a pair of strings. For both Hash1 and Hash2, the encoding is of the type where the first string is followed by the second string. Typically, the first string will be the header and the second string will be the message. The header may remain constant throughout a session while the message will change. So, it will be advantageous to process the header once and store the result.

3.3 Vector Input Hash Functions: AXU for Variable Length Vectors

The hash constructions described so far guarantee low differential probabilities only for two vectors having the same number of components. It is of interest to consider hash functions which take as input a vector of strings such that low differential probabilities are guaranteed even for two vectors having different number of components.

The generic constructions themselves are actually Hash1 and Hash2, but, the underlying encoding of a vector of strings to a single string is different. The third encoding defined in Section 3.1 is used. For single-input hash functions satisfying the null extension property or the hash function defined by the Horner's rule, Hash1 is used. For general single-input hash functions, Hash2 is used. The constructions are as follows:

•If h satisfies null extension, or, $h = \mathsf{Horner}$:

$$\mathsf{Hash1}_\tau[h, \mathcal{E}_3](A_1, \ldots, A_\kappa) = h_\tau(\mathcal{E}_3[w, \beta](A_1, \ldots, A_\kappa)).$$

•If h is a general hash function:

$$\mathsf{Hash2}_{\tau,\rho}[h,\psi,\mathcal{E}_3](A_1,\ldots,A_\kappa) = h_\tau(\mathcal{E}_3[w,\beta](A_1,\ldots,A_\kappa)) \oplus \psi_\rho(\mathsf{bin}_n(r))$$

where r is the number of β-bit blocks in $\mathcal{E}_3[w,\beta](A_1,\ldots,A_\kappa)$.

The results on differential probabilities of these constructions follow from the property of $\mathcal{E}_3$ given in Proposition 3. We state these results below. Using Proposition 3, the proofs of these results are similar to the proofs of Theorems 1, 2 and 3 and so are omitted.

Theorem 4. *Suppose the differential probabilities of* $\{h_\tau\}_\tau$ *for* equal length *strings are at most* ε. *Further, suppose that* $\{h_\tau\}_\tau$ *satisfies the null extension property. Let* $(A_1,\ldots,A_\kappa) \neq (A'_1,\ldots,A'_{\kappa'})$. *Then for every* $\alpha \in GF(2^n)$

$$\Pr[\mathsf{Hash1}_\tau[h,\mathcal{E}_3](A_1,\ldots,A_\kappa) \oplus \mathsf{Hash1}_\tau[h,\mathcal{E}_3](A'_1,\ldots,A'_{\kappa'}) = \alpha] \leq \varepsilon.$$

Here the probability is taken over the uniform random choice of τ.

Theorem 5. *Let*
$(A_1,\ldots,A_\kappa) \neq (A'_1,\ldots,A'_{\kappa'})$;
$C = \mathcal{E}_3[w,\beta](A_1,\ldots,A_\kappa)$ *and* $C' = \mathcal{E}_3[w,\beta](A'_1,\ldots,A'_{\kappa'})$;
$\mathsf{len}(C) = \ell \geq \ell' = \mathsf{len}(C')$.
Then for every $\alpha \in GF(2^n)$

$$\Pr[\mathsf{Hash1}_\tau[\mathsf{Horner},\mathcal{E}_3](A_1,\ldots,A_\kappa) \oplus \mathsf{Hash1}_\tau[\mathsf{Horner},\mathcal{E}_3](A'_1,\ldots,A'_{\kappa'}) = \alpha] \leq t/\#\mathbb{F}.$$

Here the probability is taken over the uniform random choice of τ *and* t *is the number of* β*-bit blocks in* C.

Theorem 6. *Suppose the differential probabilities of* $\{h_\tau\}_\tau$ *for* equal length *strings are at most* ε. *Let* $(A_1,\ldots,A_\kappa) \neq (A'_1,\ldots,A'_{\kappa'})$. *Then for every* $\alpha \in GF(2^n)$

$$\Pr[\mathsf{Hash2}_{\tau,\rho}[h,\psi,\mathcal{E}_3](A_1,\ldots,A_\kappa) \oplus \mathsf{Hash2}_{\tau,\rho}[h,\psi,\mathcal{E}_3](A'_1,\ldots,A'_{\kappa'}) = \alpha] \leq \varepsilon.$$

Here the probability is taken over the uniform random choice of (τ,ρ).

3.4 Vector Input Hash Function: Parallel Processing of the Components

There is a limitation of the methods described so far. In all these methods, the strings in the input vector have to be processed by the underlying hash function in a sequential manner one after the other. There is no scope for processing the strings in parallel. We next discuss how this can be done. The basic idea is to use a two-level hashing scheme [42, 40] where in the first level, the individual strings are hashed and in the second level, the outputs of the first level are hashed. Independent keys are used for the two levels.

There is a simplifying issue. The outputs of the first level are *fixed* length strings and so, the issue of tackling variable length strings at the second level is not required. So, the outputs of the first level are concatenated, suitably padded and hashed. The padding is also effected by one of the encoding methods specialised to a single-input string. One advantage of this approach is that the issue of variable number of components is automatically taken care of.

Encoding method $\mathcal{E}_1[\mathbf{w}, \beta]$ will be used with $\kappa = 1$ and so $\mathbf{w} = (w_1)$. When $\kappa = 1$, for $\mathcal{E}_1$ the value of w_1 is not used. To simplify notation, we will denote the simplified encoding method as simply $\mathcal{E}_1[\beta]$ and is defined as follows.

$$\mathcal{E}_1[\beta](A) = A||10^k. \tag{10}$$

As before, k is the minimum non-negative integer such that the length of the right hand side is a multiple of β. The expression in (10) is to be contrasted with $\mathcal{E}_3[w, \beta](A) = A||\mathsf{bin}_w(\mathsf{len}(A))||10^k$, where k is the minimum non-negative integer which makes the length of the right hand side to be a multiple of β.

We describe two-level extensions of Hash1 and Hash2. Depending on the nature of the underlying single-input hash function h, three concrete instantiations are obtained.

• h satsifies the null extension property:

$$\begin{aligned}\mathsf{Hash1}^{(2)}_{\tau,\tau'}[h, \mathcal{E}_1[\beta]](A_1, \ldots, A_\kappa) &= \mathsf{Hash1}_{\tau'}[h, \mathcal{E}_1[\beta]](\mathsf{str}_1|| \cdots ||\mathsf{str}_\kappa) \\ &= h_{\tau'}(\mathcal{E}_1[\beta](\mathsf{str}_1|| \cdots ||\mathsf{str}_\kappa))\end{aligned}$$

where for $i = 1, \ldots, \kappa$, $\mathsf{str}_i = \mathsf{Hash1}_\tau[h, \mathcal{E}_1[\beta]](A_i) = h_\tau(\mathcal{E}_1[\beta](A_i))$.

• $h = \mathsf{Horner}$:

$$\begin{aligned}\mathsf{Hash1}^{(2)}_{\tau,\tau'}[\mathsf{Horner}, \mathcal{E}_3[w, \beta]](A_1, \ldots, A_\kappa) &= \mathsf{Hash1}_{\tau'}[\mathsf{Horner}, \mathcal{E}_3[w, \beta]](\mathsf{str}_1|| \cdots ||\mathsf{str}_\kappa) \\ &= \mathsf{Horner}_{\tau'}(\mathcal{E}_3[w, \beta](\mathsf{str}_1|| \cdots ||\mathsf{str}_\kappa))\end{aligned}$$

where for $i = 1, \ldots, \kappa$, $\mathsf{str}_i = \mathsf{Hash1}_\tau[\mathsf{Horner}, \mathcal{E}_3[w, \beta]](A_i) = \mathsf{Horner}_\tau(\mathcal{E}_3[w, \beta](A_i))$.

• General h:

$$\begin{aligned}\mathsf{Hash2}^{(2)}_{\tau,\rho,\tau',\rho'}[h, \psi, \mathcal{E}_3[w, \beta]](A_1, \ldots, A_\kappa) &= \mathsf{Hash2}_{\tau',\rho'}[h, \psi, \mathcal{E}_3[w, \beta]](\mathsf{str}_1|| \cdots ||\mathsf{str}_\kappa) \\ &= h_{\tau'}(\mathcal{E}_3[w, \beta](\mathsf{str}_1|| \cdots ||\mathsf{str}_\kappa)) \oplus \psi_{\rho'}(r')\end{aligned}$$

where for $i = 1, \ldots, \kappa$, $\mathsf{str}_i = \mathsf{Hash2}_{\tau,\rho}[h, \mathcal{E}_3[w, \beta]](A_i) = h_\tau(\mathcal{E}_3[w, \beta](A_i)) \oplus \psi_\rho(r_i)$; r' is the number of β-bit blocks in $\mathcal{E}_3[w, \beta](\mathsf{str}_1|| \cdots ||\mathsf{str}_\kappa)$ and r_i is the number of β-bit blocks in $\mathcal{E}_3[w, \beta](A_i)$.

It is to be noted that for all the three constructions, the individual str_is can be computed in parallel and then concatenated at the end and hashed. In situations where such parallelism can be utilised, better performance results will be obtained.

The constructions achieve low differential probabilities for vectors with variable number of components. These follow from Theorems 1, 2 and 3. We provide the proof of Theorem 7 with the proofs of the results which follow being similar.

Theorem 7. *Suppose the differential probabilities of* $\{h_\tau\}_\tau$ *for* equal length *strings are at most* ε. *Further, suppose that* $\{h_\tau\}_\tau$ *satisfies the null extension property. Let* $(A_1, \ldots, A_\kappa) \neq (A'_1, \ldots, A'_{\kappa'})$. *Then for every* $\alpha \in GF(2^n)$

$$\Pr[\mathsf{Hash1}^{(2)}_{\tau,\tau'}[h, \mathcal{E}_1](A_1, \ldots, A_\kappa) \oplus \mathsf{Hash1}^{(2)}_{\tau,\tau'}[h, \mathcal{E}_1](A'_1, \ldots, A'_{\kappa'}) = \alpha] \leq 2\varepsilon.$$

Here the probability is taken over the independent and uniform random choices of τ *and* τ'.

Proof. Let $\mathsf{str}_i = \mathsf{Hash1}_\tau[h, \mathcal{E}_1[\beta]](A_i)$ and $\mathsf{str}'_j = \mathsf{Hash1}_\tau[h, \mathcal{E}_1[\beta]](A'_j)$. Set $\mathsf{str} = \mathsf{str}_1||\cdots||\mathsf{str}_\kappa$ and $\mathsf{str}' = \mathsf{str}'_1||\cdots||\mathsf{str}'_{\kappa'}$. We compute as follows.

$$\begin{aligned}
&\Pr[\mathsf{Hash1}^{(2)}_{\tau,\tau'}[h, \mathcal{E}_1](A_1, \ldots, A_\kappa) \oplus \mathsf{Hash1}^{(2)}_{\tau,\tau'}[h, \mathcal{E}_1](A'_1, \ldots, A'_{\kappa'}) = \alpha] \\
&= \Pr[\mathsf{Hash1}_{\tau'}[h, \mathcal{E}_1[\beta]](\mathsf{str}) \oplus \mathsf{Hash1}_{\tau'}[h, \mathcal{E}_1[\beta]](\mathsf{str}') = \alpha] \\
&= \Pr[\mathsf{str} = \mathsf{str}'] + \Pr[\mathsf{Hash1}_{\tau'}[h, \mathcal{E}_1[\beta]](\mathsf{str}) \oplus \mathsf{Hash1}_{\tau'}[h, \mathcal{E}_1[\beta]](\mathsf{str}') = \alpha | \mathsf{str} \neq \mathsf{str}'] \\
&= \Pr[\mathsf{str} = \mathsf{str}'] + \varepsilon.
\end{aligned}$$

The last inequality is over the randomness of τ' and follows from Theorem 1. We next consider the event $\mathsf{str} = \mathsf{str}'$. This event necessarily implies that $\kappa = \kappa'$.

$$\begin{aligned}
\Pr[\mathsf{str} = \mathsf{str}'] &= \Pr\left[\bigwedge_{i=1}^{\kappa} \mathsf{str}_i = \mathsf{str}'_i\right] \\
&\leq \Pr[\mathsf{str}_1 = \mathsf{str}'_1] \\
&= \Pr[\mathsf{Hash1}_\tau[h, \mathcal{E}_1[\beta]](A_1) = \mathsf{Hash1}_\tau[h, \mathcal{E}_1[\beta]](A'_1)] \\
&\leq \varepsilon.
\end{aligned}$$

The last inequality is over the randomness of τ and again follows from Theorem 1. □

Theorem 8. *Let*
$(A_1, \ldots, A_\kappa) \neq (A'_1, \ldots, A'_{\kappa'})$*;*
$C = \mathcal{E}_3[w, \beta](A_1, \ldots, A_\kappa)$ *and* $C' = \mathcal{E}_3[w, \beta](A'_1, \ldots, A'_{\kappa'})$*;*
$\mathsf{len}(C) = \ell \geq \ell' = \mathsf{len}(C')$*.*
Then for every $\alpha \in GF(2^n)$

$$\Pr[\mathsf{Hash1}^{(2)}_{\tau,\tau'}[\mathsf{Horner}, \mathcal{E}_2](A_1, \ldots, A_\kappa) \oplus \mathsf{Hash1}^{(2)}_{\tau,\tau'}[\mathsf{Horner}, \mathcal{E}_2](A'_1, \ldots, A'_{\kappa'}) = \alpha] \leq 2t/\#\mathbb{F}.$$

Here the probability is taken over the independent and uniform random choices of τ *and* τ' *and* t *is the number of* β*-bit blocks in* C*.*

Theorem 9. *Suppose the differential probabilities of* $\{h_\tau\}_\tau$ *for* equal length *strings are at most* ε*. Let* $(A_1, \ldots, A_\kappa) \neq (A'_1, \ldots, A'_{\kappa'})$*. Then for every* $\alpha \in GF(2^n)$

$$\Pr[\mathsf{Hash2}_{\Upsilon,\Upsilon'}[h, \psi, \mathcal{E}_3](A_1, \ldots, A_\kappa) \oplus \mathsf{Hash2}_{\Upsilon,\Upsilon'}[h, \psi, \mathcal{E}_3](A'_1, \ldots, A'_{\kappa'}) = \alpha] \leq 2\varepsilon.$$

Here $\Upsilon = (\tau, \rho)$*,* $\Upsilon = (\tau', \rho')$ *and the probability is taken over the independent and uniform random choices of* τ, ρ, τ' *and* ρ'*.*

3.5 A Summary of the Hash Function Constructions

The previous sections have described several types of vector-input hash functions. At a broad level, the constructions can be divided into two different groups – one group where the AXU property is guaranteed only for vectors having the same number of components while the second group ensures AXU property even for vectors having different number of components.

Encoding methods $\mathcal{E}_1$ and $\mathcal{E}_2$ are used for the first kind of construction while $\mathcal{E}_3$ is used for the second kind. Both $\mathcal{E}_1$ and $\mathcal{E}_2$ allow the maximum lengths of the individual strings to be different while for $\mathcal{E}_3$ all these maximum lengths must be equal. Further, $\mathcal{E}_1$ does not append the length

of the last string, but both $\mathcal{E}_2$ and $\mathcal{E}_3$ do so. As a result, the length of encodings using $\mathcal{E}_2$ will be the shortest while the length of encoding using $\mathcal{E}_3$ will be the longest which will in turn affect the efficiency of the hashing.

The construction of vector-input hash function is also determined to a certain extent by properties of the underlying single-input hash function. If this hash function satisfies the null extension property, then the construction is the simplest; the special and important case of hashing using Horner's rule is tackled separately and a general construction is provided when there is no condition on the single-input hash function.

Two-level hashing is used to obtain constructions where the individual components of the vectors can be processed in parallel. This eliminates some of the complexities of the encoding method. While two-level hashing provides opportunities for parallelism, the total amount of computation will be somewhat more so that a strictly sequential method for evaluating such hashes will be a little slower than the other hash functions that have been described. An advantage of using two-level hashing is that AXU property for variable length vectors is automatically achieved.

4 MAC Schemes from Stream Ciphers

As mentioned earlier, a PRF whose output consists of binary strings of some fixed length can be used as a MAC scheme. Alternatively, one can consider nonce-based MAC schemes in situations where a nonce is available. In this section, we describe constructions of nonce-based MAC schemes and fixed output length PRF from stream ciphers with IV.

4.1 Nonce-Based MAC

A well-known method [43] for constructing a MAC is to combine a PRF with a hash function in the following manner: given a nonce N and a message M, the digest is defined to be

$$\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus \mathsf{PRF}_K(N). \tag{11}$$

The secret key of the MAC algorithm is (K, τ), i.e., it consists of the secret key K of the PRF and the secret key τ of the hash function.

If $E_K : \{0,1\}^n \to \{0,1\}^n$ is the encryption function of a block cipher, then one can define a MAC scheme with key (K, τ) which takes (N, M) to $\mathsf{Hash}_\tau(M) \oplus E_K(N)$. This construction replaces the PRF in (11) by the block cipher E_K. Analysis of the block cipher based MAC has been done in [41, 9]. This analysis takes into account the fact that the block cipher is a permutation which is not true if a general PRF is used.

When using a stream cipher with IV, the PRF in (11) is replaced by SC, i.e., $\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus \mathsf{SC}_K(N)$. Here N plays the role of the IV and are n-bit strings. As long as τ is short, i.e., Hash is a Type-I hash function, the above construction is efficient and practical. For a Type-II hash function, the hash key τ is very long. Generating such a long uniform random bit string and storing it securely is very difficult. This reduces the practicability of the scheme.

A MAC scheme is shown in Table 1. This can be instantiated using either Type-I or Type-II hash function. For $\kappa = 1$, the integrity algorithm [1] of 3GPP is a variant of the scheme given in Table 1. The difference is that the values of R and τ are produced as $(\tau, R) = \mathsf{SC}_K(N)$. Compared to the scheme in Table 1, this change does not improve the efficiency or the practicability. So, we have chosen to ignore this variant.

Table 1. SC-NMAC scheme: a nonce based MAC built from a stream cipher with IV. The length of R is equal to the length of the digest of the hash function Hash. The key to the hash function is τ which could be made up of several sub-keys as discussed in Section 3. N is the nonce and $(M_1, \ldots, M_\kappa)$, $\kappa \geq 1$, is the vector of strings that is to be authenticated.

$\mathsf{SC\text{-}NMAC}_K(N, M_1, \ldots, M_\kappa)$
  $(R, \tau) = \mathsf{SC}_K(N)$;
  $\mathsf{tag} = \mathsf{Hash}_\tau(M_1, \ldots, M_\kappa) \oplus R$;
return tag.

The analysis of the security of the MAC construction (as well as the later constructions) will involve several stages of analysis of conditional probabilities. A useful fact to keep in mind while following this analysis is the following simple result.

Proposition 4. *Let E and F be events and suppose that F is the disjoint union of c other events $F_1, F_2, \ldots, F_c$ such that $\Pr[E|F_i] \leq \varepsilon$ for $i = 1, \ldots, c$. Then $\Pr[E|F]$ is also at most ε. Further, if $\Pr[F] = 1$, then $\Pr[E] \leq \varepsilon$.*

The basic idea of the construction is the following. Assume that on distinct inputs SC produces independent and uniform random strings. Then the mask R and the hash key τ can be assumed to be independent and uniform random. The hash function is used on the message vector to produce the digest. Assuming that the hash function is properly chosen, these digests will be distinct and these are masked by the independent values of the mask R. So, for distinct values of the nonces, the final tags will appear independent and uniform random to an adversary.

There are two assumptions involved in the security analysis. The first is that SC is a secure PRF and the second is that the hash function has low differential probabilities. The first assumption is computational whereas the assumption on the hash function is information theoretic. Given any concrete SC, the PRF assumption about it cannot be verified. In contrast, the low differential probability assumption on the hash function can be provably established. So, the only unproved assumption is the PRF assumption on SC.

We first consider the case where $\kappa = 1$, i.e., the hash function is a single-input function. This result then readily generalises to the case of vector input hash function where the number of components in the vector can either be fixed or be variable.

Theorem 10. *Suppose $\kappa = 1$ and $\{\mathsf{Hash}_\tau\}$ is ε-AXU with $\varepsilon \geq 1/2^n$. Then*

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{SC\text{-}NMAC}}(t, q, \sigma) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t + t', q) + \varepsilon.$$

Here t' is the time required to hash q messages of total bit length σ plus some bookkeeping time.

Proof. In the first standard reduction, SC_K is replaced by a random oracle. On distinct inputs, this oracle returns independent and uniform random strings of appropriate lengths. The standard argument is the following. Let $\mathcal{A}$ be an adversary attacking the MAC scheme. Using $\mathcal{A}$, an adversary $\mathcal{B}$ attacking the PRF-property of SC is constructed as follows. $\mathcal{B}$ has access to an oracle which is either $\mathsf{SC}_K()$ (the real oracle) or a random oracle.

$\mathcal{B}$ runs $\mathcal{A}$; if $\mathcal{A}$ makes a tag-generation query on $(N^{(i)}, M^{(i)})$, $\mathcal{B}$ queries its oracle with $N^{(i)}$ to obtain a binary sequence $S^{(i)}$ of length L of which a prefix of appropriate length is formatted to obtain $(R^{(i)}, \tau^{(i)})$. This is used to generate $\mathsf{tag}^{(i)}$ which it provides to $\mathcal{A}$. At the end $\mathcal{A}$ outputs (N, M, tag); if N is different from all the previous $N^{(i)}$'s, $\mathcal{B}$ makes one more query to its oracle on

N and uses an appropriate prefix of the output to obtain (R, τ); if $N = N^{(i)}$ for some i, then $\mathcal{B}$ uses an appropriate prefix of the previously generated $S^{(i)}$ to obtain (R, τ). The pair (R, τ) is used to regenerate the tag and compare to the tag provided by $\mathcal{A}$; if these two are equal, then $\mathcal{B}$ outputs 1, else it outputs 0. Then $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ produces a successful forgery.

The time required by $\mathcal{B}$ is the time required by $\mathcal{A}$ plus the time to compute q tags corresponding to the $q-1$ queries and the forgery attempt where the query complexity of $\mathcal{A}$ is σ bits. Additionally, $\mathcal{B}$ has to maintain some bookkeeping information about the queries. The component t' takes all these times into account.

The difference in the probabilities of $\mathcal{B}$ outputting 1 corresponding to the real and random oracles is upper bounded the PRF-advantage of SC. Let $\mathsf{succ}(\mathsf{real})$ (resp. $\mathsf{succ}(\mathsf{rnd})$) be the event that $\mathcal{A}$ is successful when $\mathcal{B}$'s oracle is SC_K (resp. the random oracle). Note that if $\mathcal{B}$'s oracle is real, then $\mathcal{A}$ is attacking the MAC-property. Then, it is standard to show the following.

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{SC\text{-}NMAC}}(t, q, \sigma) = \Pr[\mathsf{succ}(\mathsf{real})] \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t + t', q) + \Pr[\mathsf{succ}(\mathsf{rnd})].$$

The rest of the analysis is to show that $\Pr[\mathsf{succ}(\mathsf{rnd})]$ is bounded above by ε. As above, let $(R^{(i)}, \tau^{(i)})$ be the response of the random oracle on input $N^{(i)}$ and let (R, τ) be the response of the random oracle on input N, where (N, M, tag) is the forgery attempt. Recall that the constraint on nonces is that they should be distinct. Combined with the fact that the oracle is now a random one, it follows that the $(R^{(i)}, \tau^{(i)})$ are independent and uniformly distributed. The forgery attempt is (N, M, tag); if N is distinct from the previous nonces, then the (R, τ) is independent of the previous $(R^{(i)}, \tau^{(i)})$; on the other hand, if N is equal to one of the previous nonces, then this independence condition does not hold.

The probability $\Pr[\mathsf{succ}(\mathsf{rnd})]$ is the probability that $\mathsf{Hash}_\tau(M) \oplus R = \mathsf{tag}$. Let this event be E. Consider the transcript of the interaction between the adversary and the oracle. This transcript consists of concrete values for the random variables $(M^{(i)}, \mathsf{tag}^{(i)})_{i=1,\ldots,q-1}$. Since each $M^{(i)}$ and $\mathsf{tag}^{(i)}$ take values from finite sets, the set of all possible transcripts is also finite. Suppose that this number is ζ. For $j = 1, \ldots, \zeta$, let F_j be the event that the j-th transcript occurs and let $F = F_1 \vee \cdots \vee F_\zeta$. Clearly, the F_j's are disjoint events and since one of the transcripts has to occur, we have $\Pr[F] = 1$. So, using Proposition 4, if we can show that $\Pr[E|F_j] \leq \varepsilon$, then it will follow that $\Pr[E] \leq \varepsilon$. In view of this, we will now assume that $M^{(1)}, \ldots, M^{(q-1)}$ and $\mathsf{tag}^{(1)}, \ldots, \mathsf{tag}^{(q-1)}$ are arbitrary fixed (non-random) values and show that the following conditional probability, whose randomness is over $(R^{(i)}, \tau^{(i)})$, $(i = 1, \ldots, q-1)$ and (R, τ) is upper bounded by ε.

$$\Pr\left[\mathsf{Hash}_\tau(M) \oplus R = \mathsf{tag} \,\middle|\, \bigwedge_{i=1}^{q-1} \left(\mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) \oplus R^{(i)} = \mathsf{tag}^{(i)}\right)\right]. \tag{12}$$

There are two cases to consider. If N is different from each $N^{(i)}$, then by the random oracle assumption, (R, τ) is distributed uniformly and is independent of all previous $(R^{(i)}, \tau^{(i)})$. Due to the independence, the conditional probability in (12) is equal to the unconditional probability $\Pr[\mathsf{Hash}_\tau(M) \oplus R = \mathsf{tag}]$ which in turn, is equal to $1/2^n \leq \varepsilon$.

So, suppose that $N = N^{(i)}$ for some i. Since (N, M) is not equal to $(N^{(i)}, M^{(i)})$, it necessarily follows that $M \neq M^{(i)}$. Further, (R, τ) is independent of $(R^{(j)}, \tau^{(j)})$ for $j \neq i$ and the probability in (12) is equal to

$$\Pr\left[\mathsf{Hash}_\tau(M) \oplus R = \mathsf{tag} | \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) \oplus R^{(i)} = \mathsf{tag}^{(i)}\right]. \tag{13}$$

Note that $(R, \tau) = \$(N)$ and $(R^{(i)}, \tau^{(i)}) = \$(N^{(i)})$, where \$ denotes the random oracle. By assumption $N = N^{(i)}$ and so $R = R^{(i)}$. (If R and τ were generated as $(\tau, R) = \mathsf{SC}_K(N)$, then this would not have held.) Since R is uniformly distributed and is independent of $\tau^{(i)}$, it follows that

$$\Pr[R \oplus \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) = \mathsf{tag}^{(i)}] = 1/2^n. \tag{14}$$

Also, one would expect $\tau = \tau^{(i)}$. If a Type-I hash function is used, then the hash key is a fixed length string and so $\tau = \tau^{(i)}$ holds. But, for Type-II hash functions, we are working with an overloaded notation where τ (resp. $\tau^{(i)}$) is the appropriate length prefix of the hash key required for hashing the message M (resp. $M^{(i)}$). Since the lengths of M and $M^{(i)}$ can be different, we do not necessarily have $\tau = \tau^{(i)}$.

We now consider the following joint probability.

$$\Pr\left[\mathsf{Hash}_{\tau}(M) = \mathsf{tag} \oplus R, \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) = \mathsf{tag}^{(i)} \oplus R\right].$$

Assume that $M^{(i)}$ is not shorter than M. (The other case is similar.) Then the length of τ is at most the length of $\tau^{(i)}$. By the condition on Type-II hash function, $\mathsf{Hash}_{\tau}(M) = \mathsf{Hash}_{\tau^{(i)}}(M)$, i.e., hashing using a longer key does not change the digest. Given a Boolean condition ϕ, let $[\![\phi]\!]$ be 1 if ϕ is true and is 0 if ϕ is false.

$$\begin{aligned}
&\Pr\left[\mathsf{Hash}_{\tau^{(i)}}(M) = \mathsf{tag} \oplus R, \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) = \mathsf{tag}^{(i)} \oplus R\right] \\
&= \sum_{a,b} [\![(\mathsf{Hash}_a(M) = \mathsf{tag} \oplus b) \wedge (\mathsf{Hash}_a(M^{(i)}) = \mathsf{tag}^{(i)} \oplus b)]\!] \times \Pr[\tau^{(i)} = a] \times \Pr[R = b] \\
&= \frac{1}{2^n} \times \sum_{a,b} [\![(\mathsf{Hash}_a(M) = \mathsf{tag} \oplus b) \wedge (\mathsf{Hash}_a(M^{(i)}) = \mathsf{tag}^{(i)} \oplus b)]\!] \times \Pr[\tau^{(i)} = a].
\end{aligned}$$

There is no randomness in the condition $(\mathsf{Hash}_a(M) = \mathsf{tag} \oplus b) \wedge (\mathsf{Hash}_a(M^{(i)}) = \mathsf{tag}^{(i)} \oplus b)$, i.e., it either holds with probability 0 or with probability 1. For a fixed value of a, the condition holds with probability 1, if $b = \mathsf{Hash}_a(M) \oplus \mathsf{tag} = \mathsf{Hash}_a(M^{(i)}) \oplus \mathsf{tag}^{(i)}$, otherwise it holds with probability 0. So, for every a, there is at most one b for which the probability is non-zero. As a result,

$$\begin{aligned}
&\Pr\left[\mathsf{Hash}_{\tau^{(i)}}(M) = \mathsf{tag} \oplus R, \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) = \mathsf{tag}^{(i)} \oplus R\right] \\
&= \frac{1}{2^n} \times \sum_{a,b} [\![(\mathsf{Hash}_a(M) = \mathsf{tag} \oplus b) \wedge (\mathsf{Hash}_a(M^{(i)}) = \mathsf{tag}^{(i)} \oplus b)]\!] \times \Pr[\tau^{(i)} = a] \\
&\leq \frac{1}{2^n} \times \sum_{a} [\![\mathsf{Hash}_a(M) \oplus \mathsf{Hash}_a(M^{(i)}) = \mathsf{tag} \oplus \mathsf{tag}^{(i)}]\!] \times \Pr[\tau^{(i)} = a] \\
&\leq \frac{1}{2^n} \times \Pr\left[\mathsf{Hash}_{\tau^{(i)}}(M) \oplus \mathsf{Hash}_{\tau^{(i)}}(M^{(i)}) = \mathsf{tag} \oplus \mathsf{tag}^{(i)}\right] \\
&\leq \frac{\varepsilon}{2^n}.
\end{aligned}$$

Combined with (14), this shows that the probability in (13) and hence, the probability in (12) is bounded above by ε. Combined with the other inequalities, we get the desired result. □

The extension of Theorem 10 to vector input hash functions is given below.

Theorem 11. *Suppose $\kappa \geq 1$ and $\{\mathsf{Hash}_\tau\}$ is ε-AXU on κ-component vectors with $\varepsilon \geq 1/2^n$. Then*

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{SC\text{-}MAC}}(t, q, \sigma) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t + t', q) + \varepsilon.$$

Here t' is the time required to hash q messages of total bit length σ plus some bookkeeping time.

Theorem 12. *Suppose $\{\mathsf{Hash}_\tau\}$ is ε-AXU on vectors with variable number of components and $\varepsilon \geq 1/2^n$. Then*

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{SC\text{-}MAC}}(t, q, \sigma) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t + t', q) + \varepsilon.$$

Here t' is the time required to hash q messages of total bit length σ plus some bookkeeping time.

4.2 Fixed Output Length PRF

SC maps *fixed-length* bit strings to *long* (fixed-length) strings and the security assumption on SC is that of a PRF. We briefly consider how SC can be used to construct a PRF which maps *long and variable-length strings* to *short fixed-length* bit strings. Such a PRF immediately gives a MAC algorithm which does not use a nonce.

Table 2. MAC schemes based on a stream cipher with IV. The key to the hash function is τ which could have be made up of several sub-keys as discussed in Section 3.

$\mathsf{SC\text{-}MAC}_{K,\tau}(M_1, \ldots, M_\kappa)$	$\mathsf{SC\text{-}MACa}_K(M_1, \ldots, M_\kappa)$
$N = \mathsf{Hash}_\tau(M_1, \ldots, M_\kappa)$; $\mathsf{tag} = \mathsf{SC}_K(N)$; return tag.	$\tau = \mathsf{SC}_K(\mathsf{fStr})$; $N = \mathsf{Hash}_\tau(M_1, \ldots, M_\kappa)$; $\mathsf{tag} = \mathsf{SC}_K(N)$; return tag.

The idea behind the security of SC-MAC is the following. If Hash has low collision probabilities, then on distinct inputs, with high probability $N^{(i)}$'s are also distinct, where $N^{(i)} = \mathsf{Hash}_\tau(M^{(i)})$. Conditioned under this event and assuming SC_K to be a PRF, the outputs $\mathsf{tag}^{(i)}$'s of SC appear to be independent and uniformly distributed to a computationally bounded adversary.

If the hash function is ε-AU, then the probability that the $N^{(i)}$'s are distinct is at most $\binom{q}{2}\varepsilon \leq q^2\varepsilon$. Using this, we obtain the following result for the single-input hash function.

Theorem 13. *Suppose $\kappa = 1$ and $\{\mathsf{Hash}_\tau\}$ is ε-AU with $\varepsilon \geq 1/2^n$. Then*

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{SC\text{-}MAC}}(t, q, \sigma) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t + t', q) + q^2\varepsilon.$$

Here t' is the time required to hash q messages of total bit length σ plus some bookkeeping time.

Note that the requirement on the hash function is AU instead of AXU as is required in Theorem 10 for SC-NMAC. The bound in Theorem 13, on the other hand, is larger: $q^2\varepsilon$ instead of ε appearing in Theorem 10. Extensions of Theorem 13 for vector-input hash functions are straightforward and theorem statements analogous to those of Theorems 11 and 12 can be easily obtained.

The secret key for SC-MAC is the pair (K, τ). For Type-II hash functions, τ will be long and in that case it will be advantageous to generate τ using the stream cipher itself. This variant is given as SC-MACa which uses only a single key K which is the key of the underlying stream cipher. As a single key MAC scheme, SC-MACa can be used with both Type-I and Type-II hash functions.

The idea behind the security of SC-MACa is almost the same as that of SC-MAC. The only difference is that we need to argue that the probability that fStr is equal to any $N^{(i)}$ is small. This requires an additional property of hash function. For any fixed x and uniformly distributed τ, the random variable $\mathsf{Hash}_\tau(x)$ is almost uniformly distributed.

Theorem 14. *Suppose $\kappa = 1$ and $\{\mathsf{Hash}_\tau\}$ is ε-AU with $\varepsilon \geq 1/2^n$. Further, suppose that for any fixed x, uniformly distributed τ and any n-bit string α, the probability that $\mathsf{Hash}_\tau(x)$ equals α is at most μ. Then*

$$\mathsf{Adv}^{\mathrm{auth}}_{\mathsf{SC\text{-}MAC}}(t, q, \sigma) \leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t + t', q) + q^2\varepsilon + q\mu.$$

Here t' is the time required to hash q messages of total bit length σ plus some bookkeeping time.

Again extensions of Theorem 14 to obtain statements for vector-input hash functions can be done in a straightforward manner.

5 Constructions of AE Schemes

The proposed schemes for performing AE are given in Table 3. Only the encryption algorithms are described. Corresponding decryption algorithms can be easily constructed.

Table 3. AE schemes.

AE-1.$\mathsf{Encrypt}_{K,\tau}(N, M)$ $(R, Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(C) \oplus R$; return (C, tag).	AE-2.$\mathsf{Encrypt}_{K,K'}(N, M)$ $\tau = \mathsf{SC}_K(K')$; $(R, Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(C) \oplus R$; return (C, tag).	AE-2a.$\mathsf{Encrypt}_K(N, M)$ $K' = \mathsf{SC}_K(\mathsf{fStr})$; $\tau = \mathsf{SC}_K(K')$; $(R, Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(C) \oplus R$; return (C, tag).	AE-2b.$\mathsf{Encrypt}_K(N, M)$ $(R, Z, \tau) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(C) \oplus R$; return (C, tag).
AE-3.$\mathsf{Encrypt}_{K,\tau}(N, M)$ $(R, Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus R$; return (C, tag).	AE-4.$\mathsf{Encrypt}_{K,K'}(N, M)$ $\tau = \mathsf{SC}_K(K')$; $(R, Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus R$; return (C, tag).	AE-4a.$\mathsf{Encrypt}_K(N, M)$ $K' = \mathsf{SC}_K(\mathsf{fStr})$; $\tau = \mathsf{SC}_K(K')$; $(R, Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus R$; return (C, tag).	AE-4b.$\mathsf{Encrypt}_K(N, M)$ $(R, Z, \tau) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus R$; return (C, tag).

AE-1 has been suggested as a standard method of performing authenticated encryption from a stream cipher [7]. For AE-1, a Type-I hash function is suitable. This is because, for this scheme, the hash key is provided as part of the secret key of the scheme. If a Type-II hash function is desired to be used, then the hash key will be long and it will not be practical to specify this as part of the secret key. In this case, AE-2 should be used. In this scheme, the hash key is generated from K' which itself is a fixed length string. We note that AE-2 can be used with both Type-I and Type-II hash functions.

On the face of it, AE-2 may appear to be slower compared to AE-1, since it requires two invocations of SC. There are two points to note. First, the two invocations of SC can be done in parallel. So, in hardware, if a Type-II hash function requires smaller space than a Type-I hash function, then it may be advantageous to use AE-2 in comparison to AE-1. Second, the hash key τ in AE-2 does not depend on the message. So, subject to the availability of secure storage a sufficiently

long hash key may be generated and stored once per session. Then the time for generating the hash key is amortised over all the messages hashed in one session. This approach can be used when messages are comparatively short which is true, for example, in internet communications. There are, of course, many related implementation issues such as cache misses which would also determine the actual speed.

Schemes AE-3 and AE-4 hash the message M instead of the ciphertext C. One possible advantage of hashing the message M instead of the ciphertext C is that the task of hashing and ciphertext generation can proceed in parallel. For hardware implementations, this can be an advantage.

In AE-2 and AE-4, there are two keys K and K'. These schemes can be modified to AE-2a, AE-2b and AE-4a, AE-4b respectively to obtain single key schemes. Explanations for the modified schemes are as follows.

1. AE-2a and AE-4a: In these two schemes, K' is generated by applying SC_K on a fixed n-bit string fStr as $K' = \mathsf{SC}_K(\mathsf{fStr})$. This will have a minimal effect on the security bound. But, on the other hand, it will require a separate application of SC. It should be noted, though, that K' can be generated once per session.
2. AE-2b and AE-4b: These schemes eliminate K' altogether and instead derive the hash key from the invocation of SC on N. This reduces the number of SC invocations by one, but, there is a disadvantage. The hash key is generated *after* Z and so the hashing step cannot be carried out in parallel with the encryption. While this may not be an issue in software implementation, it will definitely prove to be inefficient for hardware implementations.

Privacy of the schemes. Showing privacy is easy. Consider AE-1. Under the assumption that SC is a PRF, the privacy of the scheme is easy to argue. Suppose the q queries are $(N^{(i)}, M^{(i)})$ and the corresponding responses are $(C^{(i)}, \mathsf{tag}^{(i)})$. The nonce restriction implies that $N^{(1)}, \ldots, N^{(q)}$ are distinct. Since $N^{(1)}, \ldots, N^{(q)}$ are distinct, by the PRF assumption on SC, a computationally bounded adversary will not be able to distinguish between $\mathsf{SC}(N^{(1)}), \ldots, \mathsf{SC}(N^{(q)})$ and independent and uniform random strings of corresponding lengths.

The strings $(R^{(1)}, Z^{(1)}), \ldots, (R^{(q)}, Z^{(q)})$ are prefixes of appropriate lengths of these strings. Since $C^{(i)} = Z^{(i)} \oplus M^{(i)}$ and $\mathsf{tag}^{(i)} = \mathsf{Hash}_\tau(C^{(i)}) \oplus R^{(i)}$, the random variables

$$(C^{(1)}, \mathsf{tag}^{(1)}), \ldots, (C^{(q)}, \mathsf{tag}^{(q)})$$

also appear independent and uniformly distributed to a computationally bounded adversary. The argument for the privacy of AE-3 is the same. For schemes AE-2 and AE-4, it is required to take into account the event that one of the nonces is equal to K'. Since K' is chosen to be a uniform random n-bit string and there are q nonces, the probability of this event is at most $q/2^n$. For the modifications of AE-2 and AE-4, this reasoning needs slight modifications.

Formalising the above argument shows the following bounds.

Theorem 15.

$$\begin{aligned}
\mathsf{Adv}^{\text{ae-priv}}_{\mathsf{AE\text{-}1}}(t, q, \sigma) &\leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', q).\\
\mathsf{Adv}^{\text{ae-priv}}_{\mathsf{AE\text{-}3}}(t, q, \sigma) &\leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', q).\\
\mathsf{Adv}^{\text{ae-priv}}_{\mathsf{AE\text{-}2}}(t, q, \sigma) &\leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', q + 1) + \frac{q}{2^n}.\\
\mathsf{Adv}^{\text{ae-priv}}_{\mathsf{AE\text{-}4}}(t, q, \sigma) &\leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', q + 1) + \frac{q}{2^n}.
\end{aligned}$$

$$\mathsf{Adv}^{\text{ae-priv}}_{\mathsf{AE\text{-}2a}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+2) + \frac{q}{2^n}.$$
$$\mathsf{Adv}^{\text{ae-priv}}_{\mathsf{AE\text{-}4a}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+2) + \frac{q}{2^n}.$$
$$\mathsf{Adv}^{\text{ae-priv}}_{\mathsf{AE\text{-}2b}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q).$$
$$\mathsf{Adv}^{\text{ae-priv}}_{\mathsf{AE\text{-}4b}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q).$$

Here t' is the time required for the following operations:

- *Hashing q strings of total length σ.*
- *Performing a total of σ XOR operations on bits.*
- *Time required for bookkeeping tasks.*

Authenticity of the schemes. In this case, the arguments for the first two schemes are slightly different with the argument for AE-2 being a little more complicated. We first present this argument below. Later, we give the argument for AE-3 from which the argument for AE-4 follows after some modifications based on the argument for AE-2.

Theorem 16. *Suppose $\{\mathsf{Hash}\}_\tau$ is ε-AXU. Then*

$$\mathsf{Adv}^{\text{ae-auth}}_{\mathsf{AE\text{-}2}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \frac{q}{2^n} + \varepsilon.$$
$$\mathsf{Adv}^{\text{ae-auth}}_{\mathsf{AE\text{-}2a}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+2) + \frac{q}{2^n} + \varepsilon.$$
$$\mathsf{Adv}^{\text{ae-auth}}_{\mathsf{AE\text{-}2b}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q) + \varepsilon.$$

Here σ includes the number of bits in the forgery attempt and t' is the time required for the following operations:

- *Hashing q strings of total length σ.*
- *Performing a total of σ XOR operations on bits.*
- *Time required for bookkeeping tasks.*

Proof. We first prove the result for AE-2. For the variants of AE-2, we indicate how the argument needs to be modified.

Given an adversary $\mathcal{A}$ attacking the authenticity of AE-2, the idea is to construct an adversary $\mathcal{B}$ which attacks the PRF-property of SC. $\mathcal{B}$ will have access to an oracle $\mathcal{O}$ which is either SC_K instantiated by a uniform random K, or it is an oracle which returns independent and uniform random strings. At the outset, $\mathcal{B}$ will choose a uniform random K' and compute $\tau = \mathcal{O}(K')$. Any encryption query made by $\mathcal{A}$ can be answered by $\mathcal{B}$ by invoking its own oracle. Finally, $\mathcal{A}$ makes a forgery attempt (N, C, tag). $\mathcal{B}$ recomputes the tag from N and C using $\mathcal{O}$ and compares to tag; if the two are equal, then $\mathcal{B}$ returns 1; else $\mathcal{B}$ returns 0.

Assume that $\mathcal{O}$ is the real oracle SC_K and suppose the queries are

$$(N^{(1)}, M^{(1)}), \ldots, (N^{(q-1)}, M^{(q-1)})$$

and the corresponding responses are $(C^{(1)}, \mathsf{tag}^{(1)}), \ldots, (C^{(q-1)}, \mathsf{tag}^{(q-1)})$. Further, suppose that the forgery attempt is (N, C, tag) with the restriction that this is not equal to $(N^{(i)}, C^{(i)}, \mathsf{tag}^{(i)})$ for any $1 \le i \le q-1$. The forgery attempt implicitly defines $(R, Z) = \mathsf{SC}_K(N)$.

Let $\mathsf{succ}(\mathsf{real})$ be the event that the forgery is accepted. Let Repeat be the event that K' is equal to one of $N^{(1)}, \ldots, N^{(q-1)}$ or N. Since K' is chosen uniformly at random, $\Pr[\mathsf{Repeat}] \leq q/2^n$. Then

$$\begin{aligned}\mathsf{Adv}^{\text{ae-auth}}_{\mathsf{AE\text{-}2}}(t,q,\sigma) &= \Pr[\mathsf{succ}(\mathsf{real})] \\ &= \Pr[\mathsf{succ}(\mathsf{real})|\mathsf{Repeat}] \times \Pr[\mathsf{Repeat}] + \Pr[\mathsf{succ}(\mathsf{real})|\overline{\mathsf{Repeat}}] \times \Pr[\overline{\mathsf{Repeat}}] \\ &\leq \Pr[\mathsf{Repeat}] + \Pr[\mathsf{succ}(\mathsf{real})|\overline{\mathsf{Repeat}}].\end{aligned}$$

Now assume that $\mathcal{O}$ is a random oracle. On distinct inputs, this oracle returns independent and uniform random strings of appropriate lengths. Let $\mathsf{succ}(\mathsf{rnd})$ be the event that the adversary is successful against the random oracle. Then, it is standard to show the following.

$$\Pr[\mathsf{succ}(\mathsf{real})|\overline{\mathsf{Repeat}}] \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t,q) + \Pr[\mathsf{succ}(\mathsf{rnd})|\overline{\mathsf{Repeat}}].$$

(The event Repeat is the essential difference between the analysis of AE-1 and AE-2. In AE-1, this event is not defined and τ is a uniform random string which is independent of all other random variables. In AE-2, under the condition $\overline{\mathsf{Repeat}}$, and assuming SC to be a PRF, we can assume that τ is a uniform random string which is independent of all other random variables. The rest of the proof is the same for both schemes.)

The analysis of $\Pr[\mathsf{succ}(\mathsf{rnd})|\overline{\mathsf{Repeat}}]$ is to show the following (with $\mathsf{SC}()$ replaced with a random oracle $\$()$).

$$\begin{aligned}&\Pr[\mathsf{succ}(\mathsf{rnd})|\overline{\mathsf{Repeat}}] \\ &= \Pr\left[\mathsf{Hash}_\tau(C) \oplus R = \mathsf{tag} \,\middle|\, \overline{\mathsf{Repeat}} \wedge \bigwedge_{i=1}^{q-1} (\mathsf{Hash}_\tau(C^{(i)}) \oplus R^{(i)} = \mathsf{tag}^{(i)})\right] \qquad (15) \\ &\leq \varepsilon.\end{aligned}$$

Strictly speaking, we should also be conditioning on the event $\bigwedge_{i=1}^{q-1}\left(Z^{(i)} = M^{(i)} \oplus C^{(i)}\right)$. But, conditioned on $\overline{\mathsf{Repeat}}$ and replacing $\mathsf{SC}()$ by $\$()$ results in both R and τ being independent of $Z^{(i)}$ for all i. So conditioning on this event will not change the probability.

There are two cases to consider. If N is different from each $N^{(i)}$, then (R, Z) is distributed uniformly and is independent of all previous $(R^{(i)}, Z^{(i)})$. Due to the independence, the conditional probability in (15) is equal to the unconditional probability $\Pr[\mathsf{Hash}_\tau(C) \oplus R = \mathsf{tag}]$ which due to the uniform distribution of R and its independence from τ, is equal to $1/2^n \leq \varepsilon$.

So, suppose that $N = N^{(i)}$ for some i. Then (R, Z) is independent of $(R^{(j)}, Z^{(j)})$ for $j \neq i$ and $R = R^{(i)}$. The probability in (15) is equal to

$$\Pr\left[\mathsf{Hash}_\tau(C) \oplus R = \mathsf{tag} | \mathsf{Hash}_\tau(C^{(i)}) \oplus R = \mathsf{tag}^{(i)}\right]. \qquad (16)$$

By the restriction on the adversary, we have $(N, C, \mathsf{tag}) \neq (N^{(i)}, C^{(i)}, \mathsf{tag}^{(i)})$. Since we are assuming $N = N^{(i)}$, it necessarily follows that $(C, \mathsf{tag}) \neq (C^{(i)}, \mathsf{tag}^{(i)})$. If $C = C^{(i)}$, then certainly $\mathsf{tag} \neq \mathsf{tag}^{(i)}$ and so the above probability is zero. So, let $C \neq C^{(i)}$. In this case, as in the proof of Theorem 10, the probability in (16) can be shown to be bounded above by ε. From this the result follows.

The argument for AE-2a is almost the same as that for AE-2. The only difference between the two schemes is that an extra stream cipher call is made to generate K'. This difference is reflected in the parameters of the PRF-advantage of SC – the number of queries to SC is $q + 1$ in the case of AE-2 while it is $q + 2$ in the case of AE-2a.

The argument for AE-2b is a straightforward modification of the argument for AE-2. The only difference in the two schemes is in the method that the hash key τ is derived. In AE-2b this is derived from the invocation of SC_K on N and there is no K'. So, the extra invocation of SC_K on K' is eliminated as is the event Repeat. The rest of the argument remains the same and gives the stated bound. □

Theorem 17. *Suppose* $\{\mathsf{Hash}\}_\tau$ *is* ε*-AXU. Then*

$$\mathsf{Adv}^{\text{ae-auth}}_{\text{AE-1}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1)+\varepsilon.$$
$$\mathsf{Adv}^{\text{ae-auth}}_{\text{AE-3}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1)+\varepsilon.$$

Here σ *and* t' *are as defined in Theorem 16.*

Proof. We give the proof for AE-3.

Suppose as before that the queries are $(N^{(1)},M^{(1)}),\ldots,(N^{(q)},M^{(q)})$ and the corresponding responses are $(C^{(1)},\mathsf{tag}^{(1)}),\ldots,(C^{(q)},\mathsf{tag}^{(q)})$. Further, suppose that the forgery attempt is (N,C,tag) with the restriction that this is not equal to $(N^{(i)},C^{(i)},\mathsf{tag}^{(i)})$ for any $1\le i\le q$.

Let $\mathsf{succ}(\mathsf{real})$ be the event that the forgery is accepted. Let SC_K be replaced by a random oracle. On distinct inputs, this oracle returns independent and uniform random strings of appropriate lengths. Let $\mathsf{succ}(\mathsf{rnd})$ be the event that the adversary is successful against the random oracle. Then, as before, it is standard to show the following.

$$\Pr[\mathsf{succ}(\mathsf{real})] \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t,q,\sigma)+\Pr[\mathsf{succ}(\mathsf{rnd})].$$

We show

$$\begin{aligned}&\Pr[\mathsf{succ}(\mathsf{rnd})]\\ &=\Pr\left[\mathsf{Hash}_\tau(M)\oplus R=\mathsf{tag}\,\middle|\,\bigwedge_{i=1}^{q}\left(Z^{(i)}=M^{(i)}\oplus C^{(i)},\mathsf{Hash}_\tau(M^{(i)})\oplus R^{(i)}=\mathsf{tag}^{(i)}\right)\right] \qquad (17)\\ &\le\varepsilon.\end{aligned}$$

In this case, we cannot ignore the event $\bigwedge_{i=1}^{q}\left(Z^{(i)}=M^{(i)}\oplus C^{(i)}\right)$. This is because the message, rather than the ciphertext is hashed. If the N in the forgery attempt is equal to some $N^{(i)}$, then Z and $Z^{(i)}$ will have an overlap. Since the adversary provides C in the forgery attempt and M equals $C\oplus Z$, this M is not independent of $Z^{(i)}$.

There are two cases to consider. If N is different from each $N^{(i)}$, then (R,Z) is distributed uniformly and is independent of all previous $(R^{(i)},Z^{(i)})$. Due to the independence, the conditional probability in (17) is equal to the unconditional probability $\Pr[\mathsf{Hash}_\tau(M)\oplus R=\mathsf{tag}]$ which in turn, is equal to $1/2^n\le\varepsilon$.

So, suppose that $N=N^{(i)}$ for some i. Then (R,Z) is independent of $(R^{(j)},Z^{(j)})$ for $j\ne i$ and $R=R^{(i)}$. As a result, (17) reduces to the following.

$$\Pr\left[\mathsf{Hash}_\tau(M)\oplus R=\mathsf{tag}\,\middle|\left(Z^{(i)}=M^{(i)}\oplus C^{(i)},\mathsf{Hash}_\tau(M^{(i)})\oplus R=\mathsf{tag}^{(i)}\right)\right]. \qquad (18)$$

Suppose that the length of M is at most the length of $M^{(i)}$, so that, Z is a prefix of $Z^{(i)}$. The other condition where $Z^{(i)}$ is a proper prefix of Z can be dealt with in a similar manner. Conditioning on the event $Z^{(i)}=M^{(i)}\oplus C^{(i)}$, fixes the value of Z. Let this value be z. Then $M=C\oplus z$. If

Z is a proper prefix of $Z^{(i)}$, then certainly $M \neq M^{(i)}$; if $Z = Z^{(i)}$, then $z = M^{(i)} \oplus C^{(i)}$ and $M = C \oplus z = C \oplus C^{(i)} \oplus M^{(i)}$.

Since $N = N^{(i)}$ and for an allowed forgery, $(N, C, \mathsf{tag}) \neq (N^{(i)}, C^{(i)}, \mathsf{tag}^{(i)})$, it necessarily follows that $(C, \mathsf{tag}) \neq (C^{(i)}, \mathsf{tag}^{(i)})$. If $C = C^{(i)}$, then $\mathsf{tag} \neq \mathsf{tag}^{(i)}$. From $M = C \oplus C^{(i)} \oplus M^{(i)}$ and $C = C^{(i)}$, it follows that $M = M^{(i)}$. Then, the probability in (18) is 0.

So, suppose that $C \neq C^{(i)}$ and then $C \oplus z = M \neq M^{(i)}$ (under the condition $Z^{(i)} = M^{(i)} \oplus C^{(i)}$). The probability in (18) is equal to

$$\Pr\left[\mathsf{Hash}_\tau(C \oplus z) \oplus R = \mathsf{tag} \,\middle|\, \mathsf{Hash}_\tau(M^{(i)}) \oplus R = \mathsf{tag}^{(i)}\right].$$

As in the proof of Theorem 10, this probability can be shown to be bounded above by ε. □

Theorem 18. *Suppose* $\{\mathsf{Hash}\}_\tau$ *is* ε*-AXU. Then*

$$\mathsf{Adv}^{\text{ae-auth}}_{\mathsf{AE\text{-}4}}(t, q, \sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', q + 1) + \frac{q}{2^n} + \varepsilon.$$
$$\mathsf{Adv}^{\text{ae-auth}}_{\mathsf{AE\text{-}4a}}(t, q, \sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', q + 2) + \frac{q}{2^n} + \varepsilon.$$
$$\mathsf{Adv}^{\text{ae-auth}}_{\mathsf{AE\text{-}4b}}(t, q, \sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', q) + \varepsilon.$$

Here σ *and* t' *are as defined in Theorem 16.*

Proof. Scheme AE-4 has similarities to both AE-2 and AE-3. Like AE-3, it hashes the message instead of the ciphertext while like AE-2, it derives the hash key from K'. Theorem 16 provides the argument for AE-2 (and its two variants) while Theorem 17 gives the argument for AE-3. A combination of these two arguments provides the proof for AE-4 (and its variants). □

6 Handling Associated Data

None of the schemes given in the previous section can directly handle associated data. It is tempting to concatenate the header and then apply the hash function. But, this does not work. To see this, first consider AE-1 and AE-2 and suppose that $\mathsf{Hash}_\tau(C)$ is replaced by $\mathsf{Hash}_\tau(H||C)$. The authentication property of this scheme is easily attacked. The attacker $\mathcal{A}$ queries the oracle on the input $(N, H = h_1, M = m_1 m_2)$ and receives $(C = c_1 c_2, \mathsf{tag})$; here h_1, m_1, m_2, c_1 and c_2 are bits and tag is computed as $\mathsf{tag} = \mathsf{Hash}_\tau(h_1 c_1 c_2) \oplus R$, where $(R, Z) = \mathsf{SC}_K(N)$. The forgery is $(N, H' = h_1 c_1, C'_2 = c_2, \mathsf{tag})$. First note that this forgery is allowed, since $H' \neq H$ (and $C' \neq C$). We have $H'||C' = H||C$, but, this is allowed. Validity of the forgery is seen as follows. Since N is not changed, R remains unchanged and so $\mathsf{Hash}_\tau(H'||C') \oplus R = \mathsf{Hash}(H||C) \oplus R = \mathsf{tag}$.

AE-3 and AE-4 also do not support associated data. Again, the simple possibility of hashing the concatenation of the header and the message (i.e., $\mathsf{Hash}_\tau(H||M)$) instead of only the message (i.e., $\mathsf{Hash}_\tau(M)$) gives rise to an insecure scheme. The adversary $\mathcal{A}$ makes a query $(N, H' = 0, M' = 00)$ and obtains in response $(C' = c'_1 c'_2, \mathsf{tag}')$. Here $\mathsf{tag}' = \mathsf{Hash}_\tau(H'||M') \oplus R = \mathsf{Hash}_\tau(000) \oplus R$. Let $(R, Z = z_1 z_2)$ be the output of $\mathsf{SC}_K(N)$ and then $c_1 = z_1$; $c_2 = z_2$. The adversary outputs $(N, H = 00, c_1, \mathsf{tag}')$ as the forgery. Since the nonce is not changed, the pair (R, Z) remains the same, and so, the single bit message $M = m_1$ corresponding to the forgery is $m_1 = c_1 \oplus z_1 = c_1 \oplus c_1 = 0$. The digest computation for the forgery is then $\mathsf{Hash}_\tau(H||M) \oplus R = \mathsf{Hash}_\tau(000) \oplus R = \mathsf{tag}'$. So, the concatenate-and-hash strategy does not work.

The requirement is a hash function which accepts two inputs and has low collision and differential probabilities. Existing hash functions can be modified to obtain such double-input hash functions as has been described in Section 3.

In Table 4, we present several ways to incorporate additional data into an AE scheme. Only the encryption algorithms are shown and the corresponding decryption algorithms can be easily constructed. The following points are to be noted regarding the different constructions.

1. Schemes described along the first column can be efficiently instantiated by Type-I hash functions, i.e., hash functions for which keys are short fixed-length bit strings. Since the hash key is part of the secret key, Type-II hash functions (where the hash key is long) are not suitable for these schemes.
2. Schemes described along the second, third and fourth columns derive the hash key τ using SC. Consequently, Type-II (as well as Type-I) hash functions can be used with these schemes.
   - The schemes in the second column have an n-bit string K' as a component of the secret key from which τ is derived as $\tau = \mathsf{SC}_K(K')$.
   - The schemes in the third column have only K as the secret key and K' is derived by applying SC to a fixed string fStr, i.e., $K' = \mathsf{SC}_K(\mathsf{fStr})$
   - The schemes in the fourth column derive the hash key τ from the application of SC to the nonce N.
3. Schemes AEAD-1 to AEAD-4 and their variants incorporate the header while hashing the ciphertext or the message. Schemes AEAD-5 to AEAD-8 and their variants, hash the header along with the nonce.
4. Schemes AEAD-1, AEAD-2, AEAD-5 and AEAD-6 hash the ciphertext, whereas Schemes AEAD-3, AEAD-4, AEAD-7 and AEAD-8 hash the message.

Among the several schemes, the proper one to choose would depend on the application requirements such as size of the secret key; use of Type-I or Type-II hash function which would in turn be determined by hardware versus software implementation; extent to which parallelism can be exploited; and the security bounds for the schemes to be obtained shortly. We desist from making an explicit recommendation, preferring instead to focus on identifying the schemes and obtaining their security analysis.

Note: In Schemes AEAD-5 to AEAD-8, the hash function Hash is used both as a double-input and as a single input hash function. This would seem to suggest that the hash function should satisfy AXU property for variable length vectors. That is not the case. What is required is that the hash function is AXU for single inputs and also for double inputs; the AXU property is not required between a single input and a double input.

The AEAD Schemes 1 to 4 are almost the same as AE schemes 1 to 4, except for the incorporation of the header information H as the first argument of the hash function. Under the assumption that the double-input hash function has low collision and differential probabilities, the analysis of these AEAD schemes is exactly the same as that of the corresponding AE schemes and hence, have the same security bounds. The same holds true for the variants of AEAD-2 and AEAD-4.

We provide explanations for Schemes 5 to 8. Since N is a nonce, the pair (H, N) is also a nonce, i.e., the freshness of N ensures the freshness of (H, N). Assuming that Hash has low collision probabilities, the outputs V are distinct and hence play the role of nonces. Conditioned on the event that the Vs are distinct, Scheme AEAD-5 reduces to Scheme AE-1; Scheme AEAD-6 reduces to Scheme AE-2; Scheme AEAD-7 reduces to Scheme AE-3; and Scheme AEAD-8 reduces to Scheme AE-4. The variants AEAD-6a and AEAD-8a are similar to earlier variants in the way they derive K'

Table 4. AEAD schemes.

AEAD-1.$\mathsf{Encrypt}_{K,\tau}(H,N,M)$ $(R,Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,C) \oplus R$; return (C, tag).	AEAD-2.$\mathsf{Encrypt}_{K,K'}(H,N,M)$ $\tau = \mathsf{SC}_K(K')$; $(R,Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,C) \oplus R$; return (C, tag).	AEAD-2a.$\mathsf{Encrypt}_K(H,N,M)$ $K' = \mathsf{SC}_K(\mathsf{fStr})$; $\tau = \mathsf{SC}_K(K')$; $(R,Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,C) \oplus R$; return (C, tag).	AEAD-2b.$\mathsf{Encrypt}_K(H,N,M)$ $(R,Z,\tau) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,C) \oplus R$; return (C, tag).
AEAD-3.$\mathsf{Encrypt}_{K,\tau}(H,N,M)$ $(R,Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,M) \oplus R$; return (C, tag).	AEAD-4.$\mathsf{Encrypt}_{K,K'}(H,N,M)$ $\tau = \mathsf{SC}_K(K')$; $(R,Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,M) \oplus R$; return (C, tag).	AEAD-4a.$\mathsf{Encrypt}_K(H,N,M)$ $K' = \mathsf{SC}_K(\mathsf{fStr})$; $\tau = \mathsf{SC}_K(K')$; $(R,Z) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,M) \oplus R$; return (C, tag).	AEAD-4b.$\mathsf{Encrypt}_K(H,N,M)$ $(R,Z,\tau) = \mathsf{SC}_K(N)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(H,M) \oplus R$; return (C, tag).
AEAD-5.$\mathsf{Encrypt}_{K,\tau}(H,N,M)$ $V = \mathsf{Hash}_\tau(H,N)$; $(R,Z) = \mathsf{SC}_K(V)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(C) \oplus R$; return (C, tag).	AEAD-6.$\mathsf{Encrypt}_{K,K'}(H,N,M)$ $(\tau_1,\tau_2) = \mathsf{SC}_K(K')$; $V = \mathsf{Hash}_{\tau_1}(H,N)$; $(R,Z) = \mathsf{SC}_K(V)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_{\tau_2}(C) \oplus R$; return (C, tag).	AEAD-6a.$\mathsf{Encrypt}_K(H,N,M)$ $K' = \mathsf{SC}_K(\mathsf{fStr})$; $(\tau_1,\tau_2) = \mathsf{SC}_K(K')$; $V = \mathsf{Hash}_{\tau_1}(H,N)$; $(R,Z) = \mathsf{SC}_K(V)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_{\tau_2}(C) \oplus R$; return (C, tag).	
AEAD-7.$\mathsf{Encrypt}_{K,\tau}(H,N,M)$ $V = \mathsf{Hash}_\tau(H,N)$; $(R,Z) = \mathsf{SC}_K(V)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_\tau(M) \oplus R$; return (C, tag).	AEAD-8.$\mathsf{Encrypt}_{K,K'}(H,N,M)$ $(\tau_1,\tau_2) = \mathsf{SC}_K(K')$; $V = \mathsf{Hash}_{\tau_1}(H,N)$; $(R,Z) = \mathsf{SC}_K(V)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_{\tau_2}(M) \oplus R$; return (C, tag).	AEAD-8a.$\mathsf{Encrypt}_K(H,N,M)$ $K' = \mathsf{SC}_K(\mathsf{fStr})$; $(\tau_1,\tau_2) = \mathsf{SC}_K(K')$; $V = \mathsf{Hash}_{\tau_1}(H,N)$; $(R,Z) = \mathsf{SC}_K(V)$; $C = M \oplus Z$; $\mathsf{tag} = \mathsf{Hash}_{\tau_2}(M) \oplus R$; return (C, tag).	

by invoking SC_K on fStr. Consequently, the bounds for these variants follow easily from the bounds for AEAD-6 and AEAD-8.

The privacy and authenticity bounds for these AEAD schemes are the same as that for the corresponding AE schemes plus an additive term of $\varepsilon \times \binom{q}{2} \leq \varepsilon \times q^2$ which accounts for the probability that there is a collision among the Vs.

Formalising the above arguments gives the following two results.

Theorem 19.

$$\begin{aligned}
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-1}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q).\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-3}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q).\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-2}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q) + \frac{q}{2^n}.\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-4}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q) + \frac{q}{2^n}.\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-2a}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \frac{q}{2^n}.\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-4a}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \frac{q}{2^n}.\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-2b}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q) + \frac{q}{2^n}.\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-4b}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q) + \frac{q}{2^n}.
\end{aligned}$$

Further, suppose $\{\mathsf{Hash}\}_\tau$ *is* ε*-AXU. Then*

$$\begin{aligned}
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-5}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q) + \varepsilon q^2.\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-7}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q) + \varepsilon q^2.\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-6}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q) + \frac{q}{2^n} + \varepsilon q^2.\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-8}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q) + \frac{q}{2^n} + \varepsilon q^2.\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-6a}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \frac{q}{2^n} + \varepsilon q^2\\
&\mathsf{Adv}^{\text{aead-priv}}_{\text{AEAD-8a}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \frac{q}{2^n} + \varepsilon q^2.
\end{aligned}$$

Here t' *is the time required for the following operations:*

- *Hashing* q *pairs of strings of total length* σ*.*
- *Performing a total of* σ *XOR operations on bits.*
- *Time required for bookkeeping tasks.*

Theorem 20. *Suppose* $\{\mathsf{Hash}\}_\tau$ *is* ε*-AXU. Then*

$$\begin{aligned}
&\mathsf{Adv}^{\text{aead-auth}}_{\text{AEAD-1}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \varepsilon.\\
&\mathsf{Adv}^{\text{aead-auth}}_{\text{AEAD-3}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \varepsilon.\\
&\mathsf{Adv}^{\text{aead-auth}}_{\text{AEAD-2}}(t,q,\sigma) \leq \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \frac{q}{2^n} + \varepsilon.
\end{aligned}$$

$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD\text{-}4}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \frac{q}{2^n} + \varepsilon.$$
$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD\text{-}5}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \varepsilon q^2 + \varepsilon.$$
$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD\text{-}7}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \varepsilon q^2 + \varepsilon.$$
$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD\text{-}6}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \frac{q}{2^n} + \varepsilon q^2 + \varepsilon.$$
$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD\text{-}8}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+1) + \frac{q}{2^n} + \varepsilon q^2 + \varepsilon.$$
$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD\text{-}2a}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+2) + \frac{q}{2^n} + \varepsilon.$$
$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD\text{-}4a}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+2) + \frac{q}{2^n} + \varepsilon.$$
$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD\text{-}6a}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+2) + \frac{q}{2^n} + \varepsilon q^2 + \varepsilon.$$
$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD\text{-}8a}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q+2) + \frac{q}{2^n} + \varepsilon q^2 + \varepsilon.$$
$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD\text{-}2b}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q) + \frac{q}{2^n} + \varepsilon.$$
$$\mathsf{Adv}^{\text{aead-auth}}_{\mathsf{AEAD\text{-}4b}}(t,q,\sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t+t',q) + \frac{q}{2^n} + \varepsilon.$$

Here σ includes the number of bits in the forgery attempt and t' is as defined in Theorem 19.

7 Deterministic Authenticated Encryption

In Table 5, we present DAEAD schemes. The basic idea behind the constructions is simple and is based on the SIV construction in [33].

The scheme DAEAD-1 in Table 5 is suitable for Type-I hash functions, i.e., hash functions where the key τ is a short fixed length string. Scheme DAEAD-2 and its variant DAEAD-2a are suitable for both Type-I and Type-II hash functions.

Table 5. DAEAD scheme.

$\mathsf{DAEAD\text{-}1}_{K,\tau}.\mathsf{Encrypt}(H,M)$	$\mathsf{DAEAD\text{-}2}_{K,K'}.\mathsf{Encrypt}(H,M)$	$\mathsf{DAEAD\text{-}2a}_{K}.\mathsf{Encrypt}(H,M)$
$V = \mathsf{Hash}_\tau(H,M);$	$\tau = \mathsf{SC}_K(K');$	$K' = \mathsf{SC}_K(\mathsf{fStr});$
$\mathsf{tag} = \mathsf{SC}_K(V);$	$V = \mathsf{Hash}_\tau(H,M);$	$\tau = \mathsf{SC}_K(K');$
$Z = \mathsf{SC}_K(\mathsf{tag});$	$\mathsf{tag} = \mathsf{SC}_K(V);$	$V = \mathsf{Hash}_\tau(H,M);$
$C = M \oplus Z;$	$Z = \mathsf{SC}_K(\mathsf{tag});$	$\mathsf{tag} = \mathsf{SC}_K(V);$
return (C, tag).	$C = M \oplus Z;$	$Z = \mathsf{SC}_K(\mathsf{tag});$
	return (C, tag).	$C = M \oplus Z;$
		return (C, tag).

Structure of the header: We have assumed the header to be a single binary string. Depending on the application, the header can have a richer structure. In general, it can be a vector of strings. Further, the number of components in the vector can be fixed or can be variable. Such richness in the syntax of the header can be handled by using an appropriate hash function. If the header is a vector of strings where the number of components in the vector is fixed, then the hash function needs to be AXU only for fixed length vectors; if, on the other hand, the number of components in the

vector can vary, then the hash function needs to be AXU for variable length vectors. Constructions of such hash functions from single-input hash functions have been described in Section 3. To handle headers with complex structures, the schemes in Table 5 have to be instantiated with appropriate hash functions. With this done, the security analysis remains unchanged.

As in the case of AE(AD) schemes, only the encryption algorithms are described in Table 5. The corresponding decryption algorithms are readily obtained. We provide a brief description of the decryption algorithm for DAEAD-1. On input (H, C, tag), compute $Z = \mathsf{SC}_K(\mathsf{tag})$; $M = C \oplus Z$; $V = \mathsf{Hash}_\tau(H, M)$; $\mathsf{ttag} = \mathsf{SC}_K(V)$; return M if $\mathsf{tag} = \mathsf{ttag}$; else return $\perp$. The quantity ttag is determined from (H, C, tag) (and the key (K, τ)); denote by $\mathsf{RGen}(H, C, \mathsf{tag})$ the value of ttag.

If the pairs (H, M) are distinct, then the outputs V of the hash function are also distinct (under the assumption that the hash function has low collision probabilities). Assuming SC to be a PRF and applying it on distinct values of V ensures that the different values of tag are independent and uniformly distributed. In particular, with high probability, these values are also distinct. Again applying SC on the different values of tag ensures that the values of Z are independent and uniformly distributed. Since C is obtained by XORing Z and M, the values of C are also independent and uniformly distributed. So, the different (C, tag) pairs are independent and uniformly distributed strings. From this the privacy of the scheme follows. Formalising this argument is routine and gives the following result.

Theorem 21. *Suppose* $\{\mathsf{Hash}\}_\tau$ *is* ε*-AU. Then*

$$\mathsf{Adv}^{\text{daead-priv}}_{\text{DAEAD-1}}(t, q, \sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', 2q) + q^2\varepsilon + \frac{q^2}{2^n}.$$
$$\mathsf{Adv}^{\text{daead-priv}}_{\text{DAEAD-2}}(t, q, \sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', 2q) + q^2\varepsilon + \frac{q^2}{2^n}.$$
$$\mathsf{Adv}^{\text{daead-priv}}_{\text{DAEAD-2a}}(t, q, \sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', 2q) + q^2\varepsilon + \frac{q^2}{2^n}.$$

Here t' is the time required for the following operations:

- *Hashing q pairs of strings of total length σ.*
- *Performing a total of σ XOR operations on bits.*
- *Time required for bookkeeping tasks.*

We next consider the authentication property. Write $\mathbf{E}_{K,\tau}$ as a shorthand for the encryption function. Note that ttag computed in the decryption module is a function of (H, C, tag) and write $\mathsf{RGen}(H, C, \mathsf{tag})$ to denote this function, i.e., $\mathsf{ttag} = \mathsf{RGen}(H, C, \mathsf{tag})$.

Theorem 22. *Suppose* $\{\mathsf{Hash}\}_\tau$ *is* ε*-AXU. Then*

$$\mathsf{Adv}^{\text{daead-auth}}_{\text{DAEAD-1}}(t, q, \sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', 2q) + q\varepsilon + \frac{1}{2^n}.$$
$$\mathsf{Adv}^{\text{daead-auth}}_{\text{DAEAD-2}}(t, q, \sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', 2q) + q\varepsilon + \frac{1}{2^n}.$$
$$\mathsf{Adv}^{\text{daead-auth}}_{\text{DAEAD-2a}}(t, q, \sigma) \le \mathsf{Adv}^{\text{prf}}_{\mathsf{SC}}(t + t', 2q) + q\varepsilon + \frac{1}{2^n}.$$

Here σ includes the number of bits in the forgery attempt and t' is as defined in Theorem 21.

Proof. We consider DAEAD-1 with the arguments for the other schemes being similar.

Suppose the queries to the encryption oracle are $(H^{(1)}, M^{(1)}), \ldots, (H^{(q-1)}, M^{(q-1)})$ and the corresponding outputs are $(C^{(1)}, \mathsf{tag}^{(1)}), \ldots, (C^{(q-1)}, \mathsf{tag}^{(q-1)})$. Let the forgery attempt be (H, C, tag).

Before getting into the detailed analysis, we make a few simple but important observations.

1. For fixed values of K and τ, the pair (C, tag) is functionally determined from (H, M). If (C', tag') is different from (C, tag), then the corresponding (H', M') is necessarily different from (H, M).
2. The forgery attempt (H, C, tag) implicitly defines a message M as $M = C \oplus \mathsf{SC}_K(\mathsf{tag})$.
3. By the restriction on the adversary, (H, C, tag) cannot be equal to $(H^{(i)}, C^{(i)}, \mathsf{tag}^{(i)})$ for any i.

If (C, tag) equals $(C^{(i)}, \mathsf{tag}^{(i)})$ for some i, then since the forgery (H, C, tag) cannot be equal to any previous $(H^{(i)}, C^{(i)}, \mathsf{tag}^{(i)})$, certainly $H \neq H^{(i)}$. If on the other hand, (C, tag) is not equal to any $(C^{(i)}, \mathsf{tag}^{(i)})$, then M is different from $M^{(i)}$. So, in both cases, (H, M) is different from all previous $(H^{(i)}, M^{(i)})$ that have been queried to the encryption oracle.

Let Seen be the following event.

$$\mathbf{E}_{K,\tau}(H^{(1)}, M^{(1)}) = (C^{(1)}, \mathsf{tag}^{(1)}), \ldots, \mathbf{E}_{K,\tau}(H^{(1)}, M^{(1)}) = (C^{(q)}, \mathsf{tag}^{(q)}).$$

Since the sources of randomness are K and τ, Seen is equivalent to the event

$$\bigwedge_{i=1}^{q} \left(Z^{(i)} = M^{(i)} \oplus C^{(i)} \right) \wedge \bigwedge_{i=1}^{q} \left(\mathsf{tag}^{(i)} = \mathsf{SC}_K(\mathsf{Hash}_\tau(H^{(i)}, M^{(i)})) \right).$$

Recall that $\mathsf{RGen}(H, C, \mathsf{tag})$ denotes the quantity ttag that is generated during decryption and compared to tag. We show that $\Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}]$ is small, which proves the authenticity of the scheme.

Note that $V^{(i)} = \mathsf{Hash}_\tau(H^{(i)}, M^{(i)})$. Since (H, M) is distinct from all previous $(H^{(i)}, M^{(i)})$, the probability that V is equal to one of the $V^{(i)}$s is at most $q\varepsilon$. Let $\mathsf{DistinctV}$ be the event that V is distinct from all the $V^{(i)}$s.

$$\Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}] \leq q\varepsilon + \Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}, \mathsf{DistinctV}].$$

In the next step, replace SC_K by a random oracle, i.e., the $\mathsf{tag}^{(i)}$s and the $Z^{(i)}$s are strings of appropriate lengths drawn independently and uniformly at random. Let this change be denoted by $\mathsf{SC2rnd}$ and for notational simplicity, we will write the corresponding probability as $\Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}, \mathsf{DistinctV}, \mathsf{SC2rnd}]$. As is standard, it is possible to show the following.

$$\begin{aligned} &\Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}, \mathsf{DistinctV}] \\ &\leq \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{SC}}(t, q, \sigma) + \Pr[\mathsf{RGen}(H, C, \mathsf{tag}) = \mathsf{tag} | \mathsf{Seen}, \mathsf{DistinctV}, \mathsf{SC2rnd}]. \end{aligned}$$

Under the condition $\mathsf{DistinctV}$ and $\mathsf{SC2rnd}$, ttag is uniformly distributed and independent of all other random variables. As a result, the probability that this value is equal to tag is $1/2^n$.

Putting the inequalities together gives the result. □

8 Conclusion

In this paper, we considered the problem of utilising a stream cipher with IV to construct several important cryptographic primitives. Several constructions are given for MAC, AE, AEAD and

DAE(AD) schemes. These constructions can be instantiated using known stream ciphers. Additionally, certain types of hash functions are required. We define suitable variants of well-known hash functions which are tailored for our applications. As a result of our work, a designer of practical cryptographic systems gets the option of using a wider variety of secure and efficient constructions for important cryptographic tasks than were previously known.

References

1. Document 1: Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3: 128-EEA3 & 128-EIA3 specification. `http://gsmworld.com/our-work/programmes-and-initiatives/fraud-and-security/gsm_security_algorithms.htm`, 28 May, 2011.
2. eSTREAM, the ECRYPT Stream Cipher Project. `http://www.ecrypt.eu.org/stream/`.
3. Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of grain-128 with optional authentication. *International Journal of Wireless and Mobile Computing*, 5(1):48–59, 2011.
4. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Okamoto [26], pages 531–545.
5. Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Okamoto [26], pages 317–330.
6. Côme Berbain and Henri Gilbert. On the security of IV dependent stream ciphers. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 254–273. Springer, 2007.
7. Daniel J. Bernstein. Cycle counts for authenticated encryption. Workshop Record of SASC 2007: The State of the Art of Stream Ciphers, dated November 1, 2007, `http://cr.yp.to/papers.html#aecycles`.
8. Daniel J. Bernstein. The Poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2005.
9. Daniel J. Bernstein. Stronger security bounds for Wegman-Carter-Shoup authenticators. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 164–180. Springer, 2005.
10. Daniel J. Bernstein. Polynomial evaluation and message authentication, 2007. `http://cr.yp.to/papers.html#pema`.
11. John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: Fast and secure message authentication. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 216–233. Springer, 1999.
12. Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.
13. Debrup Chakraborty and Palash Sarkar. A general construction of tweakable block ciphers and different modes of operations. *IEEE Transactions on Information Theory*, 54(5):1991–2006, 2008.
14. Morris Dworkin. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC, November 2011. NIST Special Publication 800-38D, `csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf`.
15. Patrik Ekdahl and Thomas Johansson. A new version of the stream cipher SNOW. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2002.
16. Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, and Tadayoshi Kohno. Helix: Fast encryption and authentication in a single cryptographic primitive. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 330–346. Springer, 2003.
17. Edgar N. Gilbert, F. Jessie MacWilliams, and Neil J. A. Sloane. Codes which detect deception. *Bell System Technical Journal*, 53:405–424, 1974.
18. Virgil D. Gligor and Pompiliu Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In Mitsuru Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 92–108. Springer, 2001.
19. Shai Halevi and Hugo Krawczyk. MMH: Software message authentication in the Gbit/second rates. In Eli Biham, editor, *Fast Software Encryption*, volume 1267 of *Lecture Notes in Computer Science*, pages 172–189. Springer, 1997.
20. Tetsu Iwata and Kan Yasuda. Hbs: A single-key mode of operation for deterministic authenticated encryption. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 394–415. Springer, 2009.

21. Charanjit S. Jutla. Encryption modes with almost free message integrity. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544. Springer, 2001.
22. Jonathan Katz and Moti Yung. Complete characterization of security notions for probabilistic private-key encryption. In *STOC*, pages 245–254, 2000.
23. Theodore D. Krovetz. *Software-Optimized Universal Hashing and Message Authentication.* PhD thesis, University of California, Davis, 2000. `http://fastcrypto.org/umac/umac\_thesis.pdf`.
24. David A. McGrew and John Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.
25. Frédéric Muller. Differential attacks against the Helix stream cipher. In Roy and Meier [34], pages 94–108.
26. Tatsuaki Okamoto, editor. *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*. Springer, 2000.
27. Michael O. Rabin and Shmuel Winograd. Fast evaluation of polynomials by rational preparation. *Communications on Pure and Applied Mathematics*, 25:433–458, 1972.
28. Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 98–107. ACM, 2002.
29. Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.
30. Phillip Rogaway. Nonce-based symmetric encryption. In Roy and Meier [34], pages 348–359.
31. Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.
32. Phillip Rogaway and Don Coppersmith. A software-optimised encryption algorithm. In Ross J. Anderson, editor, *FSE*, volume 809 of *Lecture Notes in Computer Science*, pages 56–63. Springer, 1993.
33. Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.
34. Bimal K. Roy and Willi Meier, editors. *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*. Springer, 2004.
35. Palash Sarkar. A new multi-linear hash family. *Designs, Codes, and Cryptography.* to appear.
36. Palash Sarkar. A general mixing strategy for the ECB-Mix-ECB mode of operation. *Inf. Process. Lett.*, 109(2):121–123, 2008.
37. Palash Sarkar. Efficient tweakable enciphering schemes from (block-wise) universal hash functions. *IEEE Transactions on Information Theory*, 55(10):4749–4759, 2009.
38. Palash Sarkar. Pseudo-random functions and parallelizable modes of operations of a block cipher. *IEEE Transactions on Information Theory*, 56(8):4025–4037, 2010.
39. Palash Sarkar. A simple and generic construction of authenticated encryption with associated data. *ACM Trans. Inf. Syst. Secur.*, 13(4):33, 2010.
40. Palash Sarkar. A trade-off between collision probability and key size in universal hashing using polynomials. *Des. Codes Cryptography*, 58(3):271–278, 2011.
41. Victor Shoup. On fast and provably secure message authentication based on universal hashing. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 313–328. Springer, 1996.
42. Douglas R. Stinson. Universal hashing and authentication codes. *Des. Codes Cryptography*, 4(4):369–380, 1994.
43. Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.
44. Shmuel Winograd. A new algorithm for inner product. *IEEE Transactions on Computers*, 17:693–694, 1968.